НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами” (Android)

Виконав: студент 4 курсу, групи ТВ-51

Бойко Владислав Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Ковальчук Артем Михайлович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Бойко Владислава Олександровича
(прізвище, ім’я, по батькові)

1. Тема роботи “Розробка спеціалізованого інтернет сервісу “Відкритий спортмайданчик з е-сервісами”(Android)”

керівник роботи доцент, к.т.н. Ковальчук Артем Михайлович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи MAC OS, мова програмування JAVA, мова програмування KOTLIN.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації взаємодії, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень)
1. Постановка задачі. 2.Особливості розробки. 3.Специфікації вимог до мобільного застосунку “Відкритий спортмайданчик з е-сервісами”. 4.Основні інструменти для

реалізації додатку. 5.Архітектура та бібліотеки для розробки мобільного додатку. 6.Програмна реалізація Android-додатку інтернет-сервісу

Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	01.12.2018 – 31.03.2019	
2.	Розробка архітектури та загальної структури системи	01-21.04.2019	
3.	Розробка структур окремих підсистем	22-28.04.2019	
4.	Програмна реалізація системи	29.04-13.05.2019	
5.	Захист програмного продукту	17 .05.2019	
6.	Оформлення пояснювальної записки	06-01.06.2019	
7.	Передзахист	31.05.2019	
8.	Захист	17.06.2019	

Студент _____ Бойко О.П
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ковальчук А.М
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мета роботи — розробка Android-додатку спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”. Досліджено особливості подібних систем та аналогів. Для створення нативного мобільного додатку використано Android SDK та мову програмування Kotlin. При розробки було використано середовище розробки Android Studio та систему автоматичної збірки проектів Gradle.

Записка містить 81 сторінок, 28 рисунків, 2 таблиці, 3 додатки і 11 посилань.

Ключові слова: ANDROID, KOTLIN, JAVA, ANDROID SDK, СИСТЕМА БРОНЮВАННЯ, Е-СЕРВІСИ, MVP, GRADLE.

ABSTRACT

The purpose of the work is to development of Android-application of specialized Internet service "Open sports ground with e-services". The features of such systems and analogues are investigated. To create a native mobile application, the Android SDK and the Kotlin programming language are used. For development, the Android Studio development environment and the gradle Gradle system have been used.

The note contains 81 pages, 28 figures, 2 tables, 3 attachments and 11 links. KEYWORDS: ANDROID, KOTLIN, JAVA, ANDROID SDK, BOOKING SYSTEM, E-SERVICES, MVP, GRADLE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IDE — Integrated Development Environment.

XML — Extensible Markup Language.

API — Application Programming Interface.

GUI — Graphical User Interface.

HTTP — HyperText Transfer Protocol

ООП — об’єктно-орієнтоване програмування

ОС — операційна система

UI — User Interface

CSS — Cascading Style Sheets

HTML — HyperText Markup Language

ВСТУП

Велика кількість бажаючих скористатися спортивним майданчиком може привести до певних незручностей у користуванні. Це може бути або недостатня, або велика кількість людей на спортивній зоні, групи людей з різними інтересами використання, надмірне користування спортивним майданчиком та обладнанням.

Мобільний додаток Інтернет-сервісу пропонує користувачам вирішити основні проблеми з організацією та чесним розподілом часу використання. Користувачам надається доступ до списку усіх подій, створених іншими користувачами. Всі дані надаються у структурованому вигляді або у вигляді списків. Також, за необхідністю, користувач має можливість створити власну подію за його потребами.

Для створення Android-додатку були використані такі технології як Java, Kotlin, XML, Android SDK, а також додаткові бібліотеки RxJava, OkHttp, Retrofit, Dagger 2, які спрощують створення програмного продукту, та його архітектурної частини.

1 ПОСТАНОВКА ЗАДАЧІ

В наш час досить швидко розвивається ера мобільних телефонів. Кожен рік вимоги до мобільних пристроїв зростають, виробники змагаються за звання кращого флагману та оснащують їх все потужнішим апаратним забезпеченням. На сьогоднішній день можливості простих телефонів майже не існує. Простий телефон еволюціснував у смартфон, та навчився робити надзвичайно багато речей. Сучасний смартфон вміє робити фотографії та відеозаписи, працювати з документами, виходити в інтернет, виконувати функції роутера, оплачувати рахунки та багато чого іншого. Аналітики вважають, що сучасні смартфони скоро перетворяться на повноцінні компьютери, до яких можна під'єднати всю периферію. З появою стандарту зв'язку 5G (передача даних зі швидкістю до 7 Гб / сек) люди будуть відмовлятися від старих джерел інтернету, таких як Wi-Fi. До того ж, смартфон може бути досить дешевою річчю, тому їм можуть користуватися всі - від дітей до літніх людей.

В силу розвитку мобільних технологій, розвивається і напрям мобільних додатків. Велика кількість додатків спричиняє велику конкуренцію на ринку, а значить, вимоги користувачів постійно будуть збільшуватися. Люди потребують швидкого та комфортного доступу до необхідної інформації та функціоналу. Зараз велика кількість зосереджена у сфері розробки мобільних додатків, від елементарних калькуляторів до додатків з додатковою реальністю. На сьогодняшній день, мобільні додатки - це потужний інструмент не тільки для користувача, а й для самого бізнеса. Вони дозволяє компаніям створювати імідж, підтримувати бренд, допомагають впроваджувати додаткові інструменти для роботи або комунікації між підрозділами компанії.

1.1 Суть технічної проблеми.

Найціннішим ресурсом для людини є час. І всі мобільні додатки спрямовані на те, щоб зекономити людині хоча б декілька хвилин. Велика кількість бажаючих скористатися спортивним майданчиком та його інвентаром може бути причиною неорганізованості та втратою власного часу. Контроль за спортивним майданчиком може допомогти уникнути багатьох проблем. Які проблеми можуть виникнути у звичайного користувача спортивного майданчика:

1. Переповненість спортивного майданчика;
2. Недостатня кількість гравців для проведення спортивної події або іншої спільної активності;
3. Неможливість проведення певних закритих заходів, наприклад, коли команда захотіла провести тренування.
4. Через відкритість спортивного майданчика, деякі особи можуть надмірно використовувати спортивні місця та їх спортивне обладнання. Через це деякі бажаючі не зможуть скористатися майданчиком та його спортивним обладнанням;
5. Якщо на спортивний майданчик прийдуть групи людей з різними інтересами, це може призвести до неможливості використання, або марного трату часу для однієї із груп людей;
6. Велика кількість людей та неорганізованість можуть призвести до того, що для гри зберуться гравці із різним рівнем навичок. Це може призвести до не комфортної гри як для тих, хто слабше грає, так і для хто набагато сильніший у активності.

За допомогою спортивного додатку користувач зможе впоратись із організаційними питаннями, такими як бронювання, створення спортивних подій та різних активностей, вибору типу активності або спортивної події, вибору місця та часу проведення спортивної події.

1.2 Аналіз та порівняльна характеристика аналогів

На українському ринку немає великої кількості спортивних додатків, які б надавали можливість користувачеві бронювати спортивні зайняття чи місця[4]. Основним конкурентом додатку є сервіс arena-booking (рисунок 1.1). Його основний функціонал:

- Можливість бронювати спортивні заняття;
- Можливість оренди спортивних майданчиків;
- Можливість орендувати клуб.

На даний момент це єдиний сервіс, який діє на території України та Києва, що надає можливість бронювати спортивні майданчики.

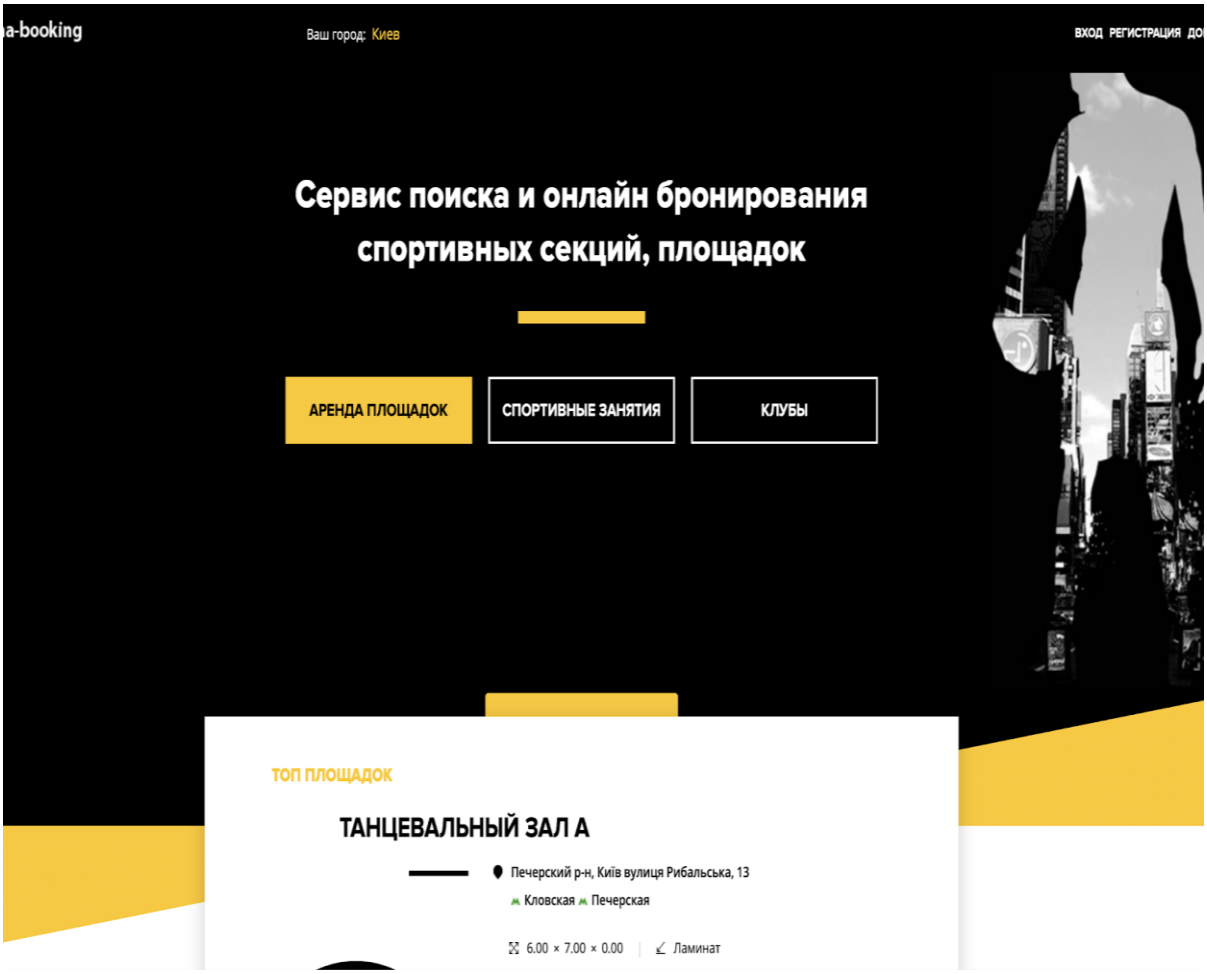

Рисунок 1.1 — Сервіс бронювання arena-booking

Але в цьому сервісі є певні недостатки:

- Неможливість створювати власні спортивні події та бронювати місце під неї;
- Існує лише WEB-додаток.

Наступним конкурентом є мобільний додаток “NEUO” (рисунок 1.2) — це програма, що дозволяє бронювати місце для занять на певних спортивних подіях. Головною особливістю цього додатку є те, що користувач має можливість фільтрувати зайняття за певним типом. Додаток має надзвичайно дружній інтерфейс, так красивий дизайн. Користувач має можливість керувати своїм профілем. Додаток має головну сторінку з певними новинами, гарячими пропозиціями щодо зайнять.

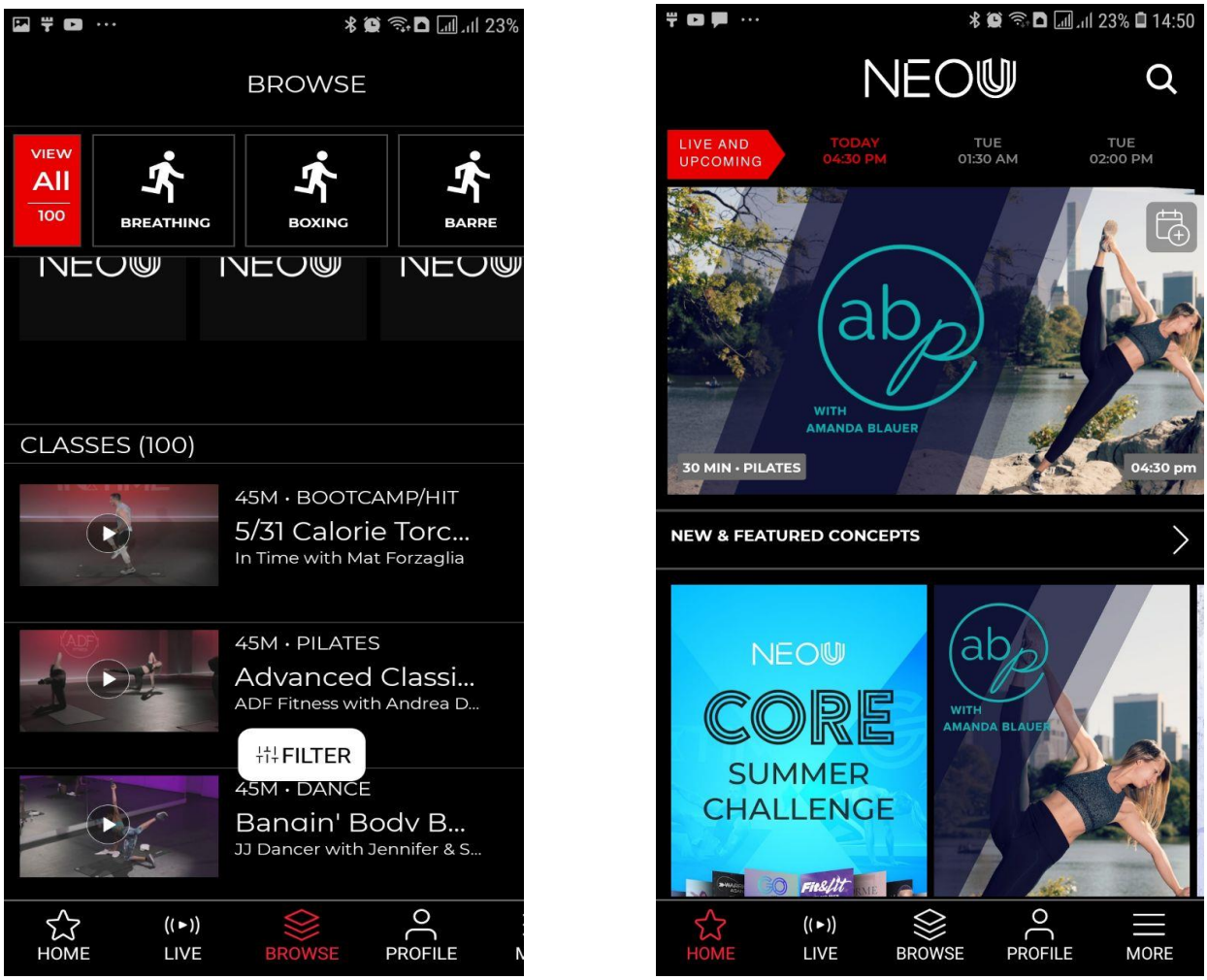

Рисунок 1.2 — мобільний додаток “NEUO”

Але цей додаток також є певні мінуси:

- Немає можливості створити власні події;
- Немає можливості забронювати цілий спортивний майданчик;
- Не працює на території України.

Також, популярним аналогом цього додатку є “MINDBODY” (рисунок 1.3) — мобільний додаток, що дозволяє бронювати не тільки місце на спортивній події, а і користуватися різними послугами такими як медітація, массажі, акупунктура. Також додаток надає користувачу можливість вести свій розклад, календар.

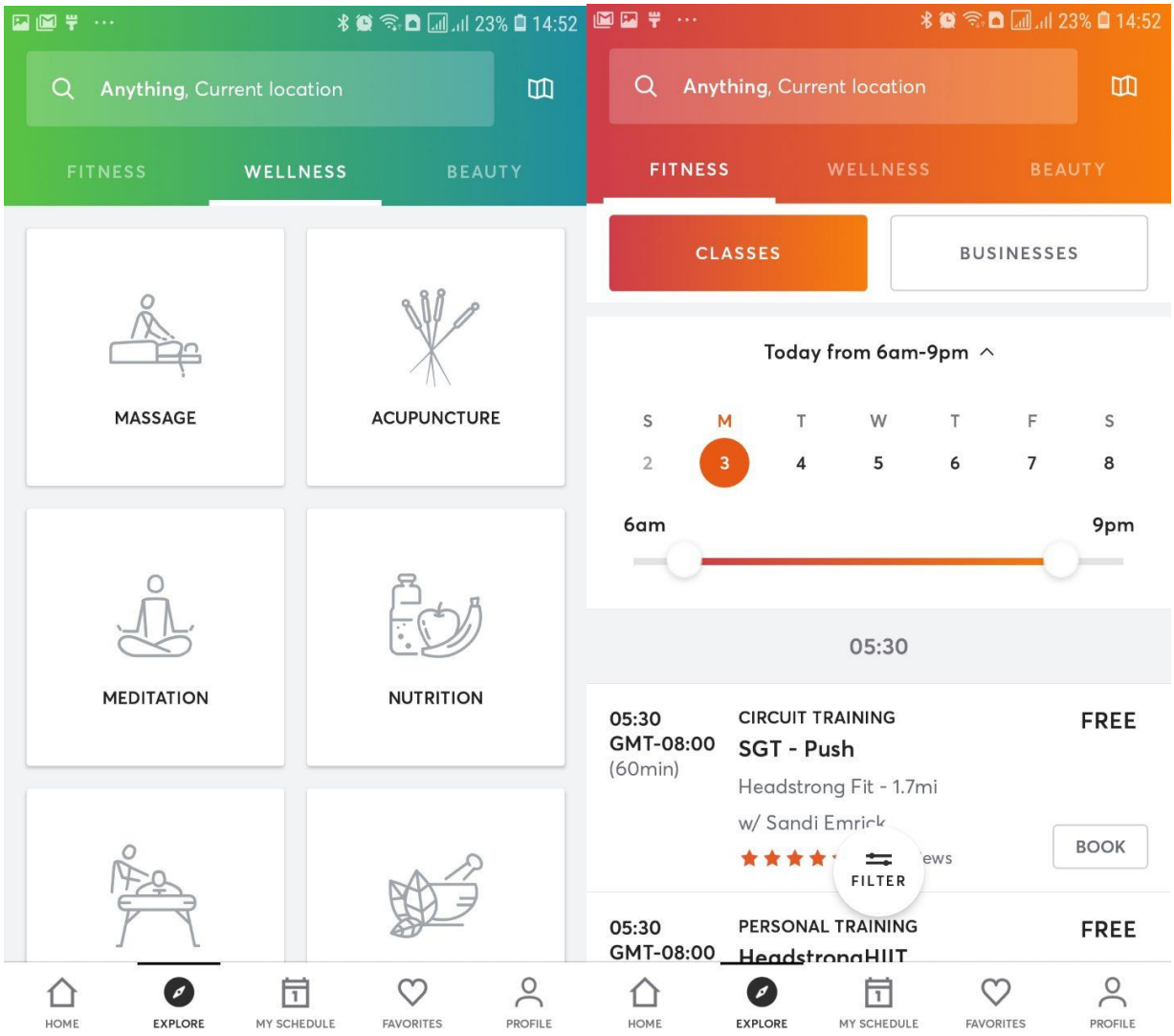

Рисунок 1.3 — мобільний додаток “MINDBODY”

Але додаток має і певні мінуси:

- Додаток працює лише у США;
- Неможливість бронювати цілий спортивний майданчик.
- Не гнучкі параметри для пошуку подій.

1.2 Мета та цілі дипломної роботи

Мета дипломної роботи — за допомогою спеціалізованого інтернет-сервісу залучити студентів до спортивного життя та спростити підхід до резерву спортивних об’єктів як у межах КПІ так я за його межами.

Для досягненні цієї мети необхідно виконати наступні завдання:

- Аналіз ринку, користувачів, які використовують даний тип мобільний додатків;
- Аналіз конкурентів та аналогів;
- Дослідити інструменти та технології для розробки мобільного додатку;
- Моделювання та проектування архітектури мобільного додатку;
- Програмна розробка мобільного додатку;

Висновки до розділу 1

В цьому розділі було розглянуто технічну проблему, що мотивує до створення програмного забезпечення. Була проведена порівняльна характеристика та аналіз подібних мобільних додатків, систем чи web-сайтів. Розглянуто мінуси подібних платформ, їх функціональність та цікаві сторони. На основі цього аналізу були поставлені цілі дипломної роботи та мету.

2 ОСОБЛИВОСТІ РОЗРОБКИ

Мобільний додаток — програмне забезпечення, що призначене для роботи на смартфонах, планшетах, та інших мобільних пристроях таких як смарт-годинники та інші. Мобільні додатки досить швидко набули свою популярність. Спочатку їх використовували для перевірок електронної пошти, але за невеликий проміжок часу їх область призначення розширилася для користування інтернетом[8], спілкування, перегляду відео, ігр для мобільних телефонів. Зараз, мобільні додатки використовують в основному на мобільних пристроях, на яких встановленні операційні системи IOS від компанії Apple та ANDROID від компанії Google.

У якості платформи для розробки мобільного додатку було вибрано ANDROID, через декілька причин, одна з них - висока популярність платформи.
Така популярність обумовлена багатьма факторами, наприклад, низька ціна. Іншим фактором є великий модельний ряд та відкритість операційної системи.

Android — операційна система для смартфонів, планшетів, електронних книг, цифрових програвачів, наручних годинників, фітнес-браслетів, ігрових приставок, ноутбуків, нетбуків, смартбуків, окулярів Google Glass, телевізорів та інших пристроїв. Ця система заснована на Linux ядрі та власній реалізації віртуальної машини Java від Google — Dalvik.

Додаток має містити дані про спортивні події, що проходять на території первних спортивних майданчиків, а також координати цих майданчиків які відображюаться на карті від Google Map Api. Використання всього функціоналу мобільного додатку може бути можливим лише після авторизації користувача. Після авторизації користувач має можливість для перегляду усіх даних які надаються серверною частиною сервісу.

2.1 Особливості розробки Android — частини сервісу

Android - найпопулярніша платформа, попит на додатки для неї високий, так само як і конкуренція між ними. Для iOS пристрої робить тільки одна компанія, а для Android - багато. Розробникам доводиться мати справу не тільки з версійна ОС, але і з пристроями, які сильно відрізняються. Один з головних каменів спотикання — розмір і роздільна здатність екрану. Android програми доводиться довше тестувати, щоб переконатися в правильності роботи на більшості смартфонів і планшетів. Ця особливість призводить до додаткових витрат часу і грошей на розробку додатків. До того ж, виникають складності при роботі з інтернетом, оскільки можуть виникати проблеми з синхронізацією довгих операцій, таких як запити в інтернет та життєвий цикл активності[6].

На сьогоднійшній день, розробники приділяють значну увагу на навантаження на батарею і пам’ять телефону. Розробники з кожною версією Android стараються якомога більше покращити продуктивність і швидкість роботи додатків, оновлюючи SDK. Усі напрями в дизайні зосереджені на простоту на для найбільшої функціональності та “юзабіліті”. Якщо раніше розробники та дизайнери старалися приділяти увагу шрифтам та прикрашуваннями додатку, то зараз усе робиться в максимально мінімалістичному та простому ключі, для зручності у користуванні.

2.2 Види мобільних застосунків

2.2.1 Мобільні сайти, веб-застосунки

Є найбільш поширеним типом додатків, призначених для мобільних пристроїв. За допомогою смартфона нинішніх поколінь можна подивитися звичайний сайт. Таким пристроїв є практично все, що ми бачимо в десктопних додатках, так як можливість підтримки HTML5 виконує своє завдання. Саме веб-додатки будуть

відмінним рішенням для початку, адже тільки завдяки ним можна за невеликі гроші і маленький термін отримати досить вагомий результат. Також перевагою мобільних сайтів в порівнянні з іншими мобільними додатками є так звана кроссплатформенність. Але серед вагомих мінусів можна відзначити, що з ними складно заробити. Веб-додатки використовують стандартні веб-технології, такі як HTML5, JavaScript і CSS. Цей підхід дає можливість створити крос-платформні мобільні додатки, які працюють на декількох пристроях[9]. Додатки на HTML5 можуть працювати на різних операційних системах і типах пристроїв. Додаток масштабується залежно від розміру пристрою[5]. Там, де необхідно оновлення, необхідно оновити і протестувати один додаток і воно буде доступно для всіх пристроїв відразу. У HTML5 додатках великою проблемою в безпеці є можливість подивитися вихідний код. Це означає, що можна не тільки зрозуміти, як воно працює, але і використовувати це в своїх цілях.

2.2.2 Гібридні додатки

З таким підходом ви отримаєте доступ до всіх переваг application programming interface (API) операційної системи, а саме — в числі приємних аспектів можна виділити те, що додаток обростає push-повідомленнями. Крім цього ваш продукт може бути розміщений в сторах. Разом з цим головний контент все ще є платформо незалежної сторінкою з версткою, яка розміщена на сервері. Завдяки цьому можна внести косметичні зміни в продукт без необхідності випускати нову версію, адже досить просто залити проведені зміни на сервер. Таким чином, гібридні програми будуть відмінним рішенням для всіх, хто хоче почати бізнес або втілити свої ідеї. Гібридні додатки поєднують в собі деякі функції нативних і веб-додатків: кроссплатформенність і можливість використання ПЗ телефону. Вони можуть бути завантажені через магазини додатків, і при цьому мають можливість незалежного оновлення інформації (вимагають підключення до інтернету, оскільки веб частина оновлюється через інтернет). Гібридні додатки сьогодні найбільш популярні, так як,

розробка відбувається швидше і дешевше, ніж у випадку з нативними додатками, хоча оболонка і написана на «рідному» мовою програмування, «начинка» може бути написана в тому чи іншому обсязі на html5. Користувач ж швидше за все не помітить різницю між нативним додатком і гібридним. HTML5 додатки, як правило, дешевше розвивати і підтримувати, ніж нативні додатки, так як це тільки один додаток, необхідне для підтримки декількох ОС. Цей додаток може бути розроблено одним веб-розробником. Гібридні додатки є портативними; можуть бути побудовані практично з тією ж швидкістю, з якою додаток HTML5, бо базова технологія та ж; можуть бути зроблені практично за тією ж вартістю, що і додаток HTML5, проте, для більшості фреймворків потрібна ліцензія, яка додає додаткові витрати на розробку; можуть бути доступні і поширені через відповідний магазин додатків, так само, як нативні додатки; мають доступ до апаратних ресурсів, як правило, за рахунок власних API. Але не всі апаратні ресурси доступні для гібридних додатків[7]. Застосунки будуть відображатися для кінцевого користувача, як нативні додатки, але працювати значно повільніше. Те ж саме обмеження на HTML5. Візуалізація складних макетів CSS займе більше часу, ніж візуалізація відповідного макета.

2.2.3 Нативні додатки

Такі додатки вважаються самим ресурсоємними, але також вони дають можливість користуватися функціями, запропонованими окремими операційними системами, в максимальному обсязі. В результаті нативні додатки займають першість серед інших видів додатків не тільки за функціональними особливостями, але також за швидкістю завантаження. Компанії, які виконують додатки комбінованого типу, приходять саме до такого підходу. Наприклад, всесвітньо відомий Facebook починав свою діяльність з додатка комбінованого типу: як контенту виступала веб-сторінка, а нативними контролю є вкладки, перемикачі та інше. Але навіть не зважаючи на те, що це рішення є непоганим, все одно

виникають проблеми з продуктивністю, тому багато розробників уникають комбінацій з Інтернетом. Оригінальне додаток — це додаток побудоване повністю з використанням технологій, що відносяться до тієї чи іншої операційної системи. Це може бути Android, IOS, Windows, Blackberry і т.д. Для Android, нативні додатки, як правило, побудовані з використанням Java, в той час як для IOS, додаток може бути побудовано з використанням Objective C або Swift. Нативні ж додатки, є унікальними для кожної операційної системи, і таким чином, щоб підтримувати декілька мобільних операційних систем, необхідно створити окремі додатки для кожної ОС. Якщо потрібне оновлення, то кожен додаток має бути оновлено незалежно один від одного. Для того щоб додаток підлаштовувалася під різні розміри пристроїв / екрану і орієнтації в процесі розробки створюються різні макети. Нативні додатки для всіх основних мобільних операційних систем, як правило, вимагають спеціалізованого розробника для кожної операційної системи (Java для Android, Objective C / Swift для IOS, C # для Windows), вартість роботи значно дорожче, ніж робота одного веб-розробника. Розробка додатків під певну платформу вимагає більше знань, ніж розробка веб-додатки на HTML5. Нативні додатки можуть взаємодіяти з широким спектром апаратних засобів, доступних на пристрої, включаючи Global Positioning System (GPS), камеру, акселерометр і багато іншого.

Висновки до розділу 2

В цьому розділі було досліджено що таке мобільний додаток, для яких операційних систем існують. Переваги, недоліки та складності розробки під Android та IOS. В цьому розділі було описані три типи мобільних додатків: нативні, гібридні та мобільні сайти. Розглянуті їх недоліки та переваги, також коротко описано методи розробки кожного типу цих додатків. Було досліджено чому нативні додатки у розробці подібних систем найрелевантніші.

3 СПЕЦИФІКАЦІЇ ВИМОГ ДО МОБІЛЬНОГО ЗАСТОСУНКУ “ВІДКРИТИЙ СПОРТМАЙДАНЧИК З Е-СЕРВІСАМИ”

3.1 Границі проекту

Завданням цього програмного є реалізувати, можливо поширити, такий тип мобільних додатків як система бронювання спортивного майданчика. Мобільний застосунок повинен виконувати контроль роботи бронювання, та роботи інших сервісів додатку, для того, щоб користувачі з середньостатистичним телефоном мав можливість скачати цей застосунок та користуватися ним, не зважаючи на апаратне забезпечення смартфону.

3.2 Загальний опис проекту, класи та характеристики користувачів

Користувачами є звичайні люди, які мають певну перепустку, що використовується для контроля. Оскільки на початковому етапі будуть використовуватися студентські квитки, то, для початку, користувачами мобільного додатку будуть звичайні студенти КПІ.

Мобільний додаток, повинен мати певну систему бронювання, те ведення профайлу користувача — студента. Система має оперувати із подіями — це тип системних даних що описує спортивну подію, яка повинна містити певну інформацію:

- Час початку події;
- Час закінчення події;

- Описання події;
- Назва події;
- Тип події (футбол, баскетбол);
- Назва спортивного майданчика;
- Координати спортивного майданчика.

Також, в системі повинна бути така сутність як користувач — “User”, що повинна мати у своєму складі такі дані:

- Електронна пошта користувача;
- Користувацьке ім’я — “Username”;
- Власне ім’я користувача;
- Фамілія користувача;

Отже, функціонал, що необхідний мобільному застосунку:

- Можливість логіну;
- Можливість реєстрації;
- Можливість переглянути власний профайл;
- Можливість вийти з додатку;
- Можливість переглянути список актуальних спортивних подій;
- Можливість переглянути події що створив користувач;
- Можливість переглянути події, до яких під’єднався користувач;
- Можливість створити власну подію;
- Можливість приєднатися до певної створеної події;
- Можливість від’єднатися від події.

3.3 Прецеденти клієнтського додатку

На основі поставленого функціоналу мобільного додатку було створено діаграму прецедентів (рисунок 3.1).

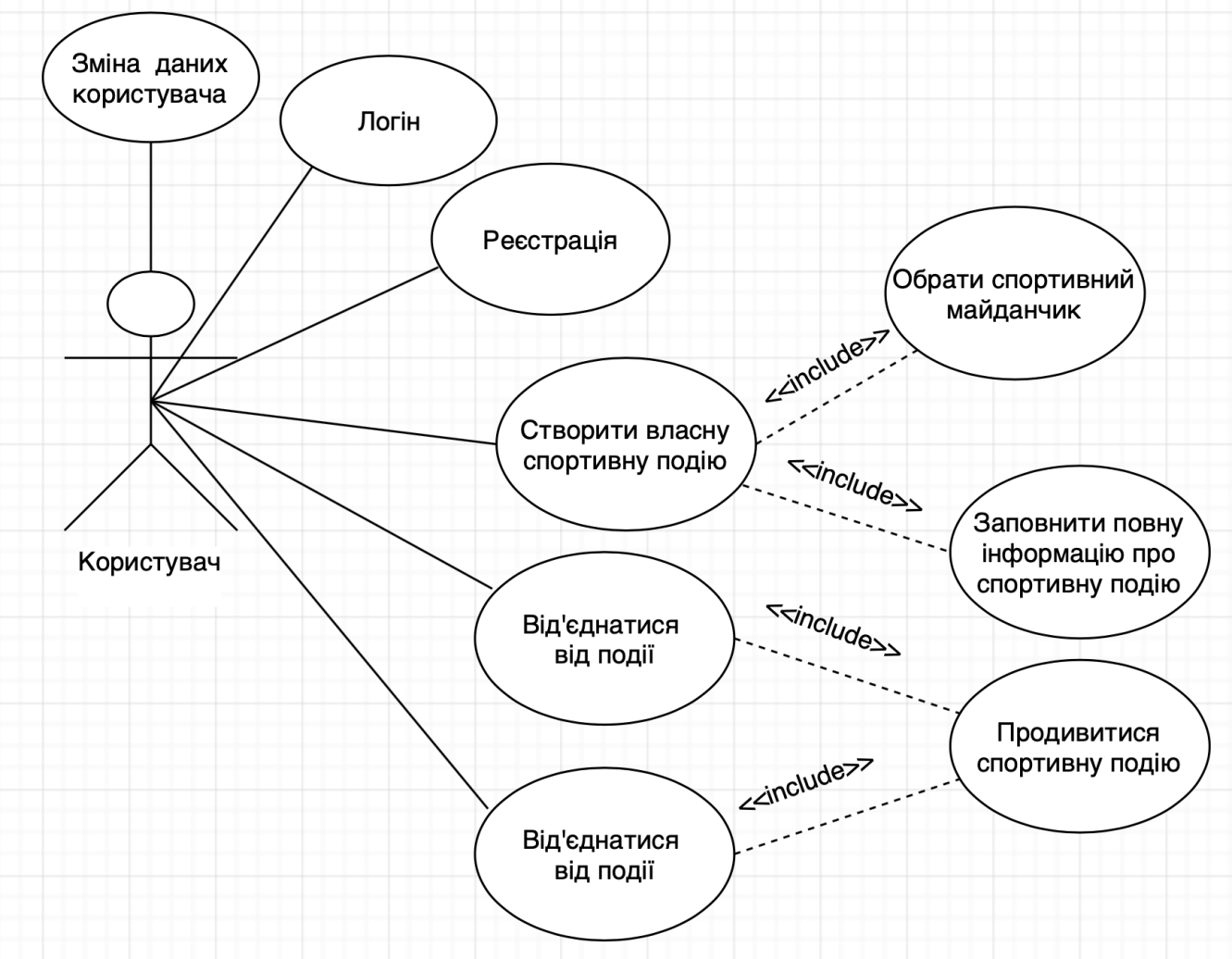

Рисунок 3.1 — Діаграма прецедентів.

Основні елементи діаграми прецедентів — учасник (actor) та прецедент (певна дія). Учасник — це набір логічно зв'язаних ролей, що відтворюються при взаємодії з попередниками або сутностями, наприклад система, користувач, сервер. Учасник може бути особою або іншою системою, підсистемою або класом, що представляє щось по суті[10]. Графічно учасник зображується як «маленька людина». Прецедент — опис набору наступних подій (включаючи варіанти), що виконуються системою, що призводить до результату, що спостерігається учасником.

3.4 Системні вимоги до додатку

Оскільки було обрано в якості операційної системи — Android, то необхідно якісно проаналізувати ринок версій цієї операційної системи, кількість користувачів, кількість девайсів, що підтримують цю операційну систему, та актуальність розробки на ній. Пошуковий гігант Google, який являється розробником цієї операційної системи надає доступ до статистичним даним по версіям. До того ж, необхідно провести статистичний аналіз пристроїв, що мають різні розміри екранів, ця інформація також надана Google.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Рисунок 3.2 — Таблиця версійності Android

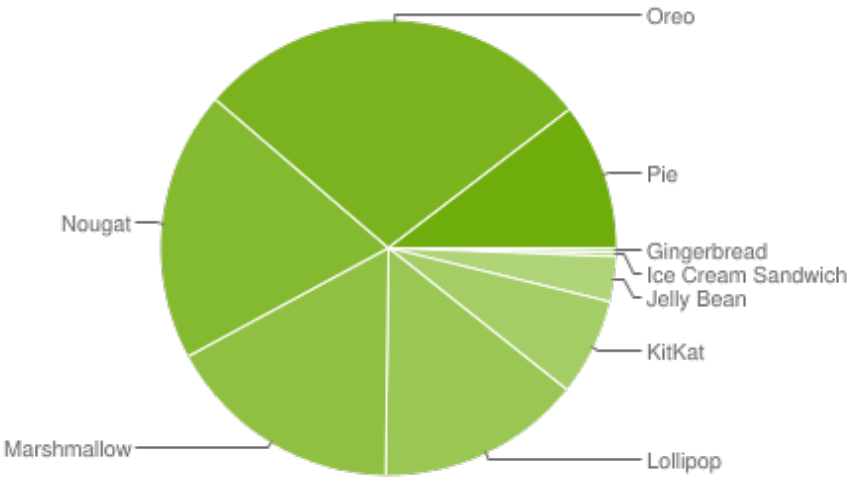

Рисунок 3.3 — Діаграма версійності Android

Судячи з таблиці та діаграми версійності (рисунок 3.3, рисунок 3.2) Android, найбільша кількість пристроїв має попередньо встановлені такі версії як: Lollipop, Marshmallow, Nougat, Oreo, Pie. Тобто, починаючи з версії Lollipop.

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	0.4%				0.1%	0.1%	0.6%
Normal		0.9%	0.3%	24.0%	37.7%	23.6%	86.5%
Large		2.4%	1.9%	0.6%	1.6%	1.7%	8.2%
Xlarge		3.1%		1.3%	0.6%		5.0%
Total	0.4%	6.4%	2.2%	25.9%	40.0%	25.4%	

Рисунок 3.4 — Таблиця екранів та їх конфігурації

Судячи з діаграми та таблиці екранів та конфігурацій (рисунок 3.4, рисунок 3.5), найбільша кількість пристроїв має у своїй конфігурації такі розміри та щільності екранів як: xhdpi, hdpi, xxhdpi.

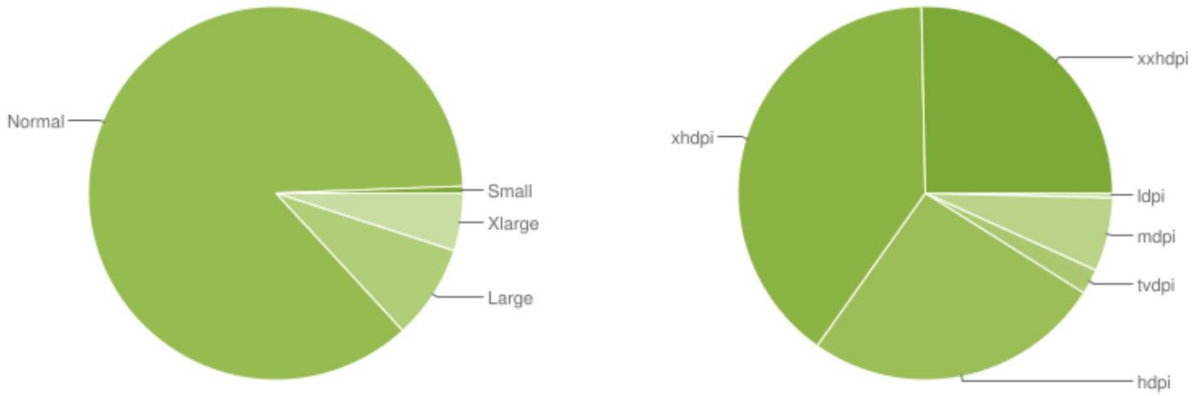

Рисунок 3.5 — Діаграма екранів та їх конфігурації

На основі наданих вище даних можна скласти вимоги для мобільного додатку:

- Операційна система: Android Lollipop 5.1 або новіша;
- Екрани зі специфікаціями xhdpi, xxhdpi, hdpi;
- Сумісність з усіма існуючими процесорними архітектурами Android;
- Мінімальний об’єм пам’яті на пристрої: 150 Мб.

Висновки до розділу 3

В цьому розділі було проведено детальний аналіз ринку мобільних пристроїв, їх специфікацій та конфігурацій. Було надано та розглянуто таблиці версійностей та таблицю конфігурацій екрану. Також додатково надано кругові діаграми версійностей та конфігурацій екрану. Побудовано діаграму прецедентів системи, яка була побудована на основі встановленого функціоналу. Розглянуті та визначені границі проекту, визначено функціонал мобільного додатку. Встановлено системні вимоги мобільного додатку за такими характеристиками: сумісність, об’єм пам’яті, специфікації екранів, операційна система, процесорна архітектура.

4 ОСНОВНІ ІНСТРУМЕНТИ ДЛЯ РЕАЛІЗАЦІЇ ДОДАТКУ

4.1 Набір бібліотек Android SDK

У процесі аналізу типів мобільних додатків було обрано розробляти нативний додаток. Розробка програм для Android зазвичай робиться з мовою програмування, подібною до Java, і набором інструментів розробки SDK (Software Development Kit), але є й інші варіанти. У своєму складі SDK містить емулятор платформи Android, він побудований на основі qemu і є дуже повільним (щонайменше). Емулятор дозволяє створювати віртуальні пристрої на різних версіях Android, на різних екранах. (Android Virtual Device або AVD у термінології SDK), де можна запускати та перевіряти створені вами програми. Емулятор дозволяє користувачеві не тільки використовувати функціонал операційної системи, а й емулювати увесь функціонал реального пристрою:

- Зміна орієнтації екрану;
- Зміна рівню заряду аккумулятора;
- Емулювати роботу камери;
- Емулювати СМС та дзвінки на телефон;
- Емулювати роботу сенсора відбитків пальців;
- Робити записи екрану;
- Емулювати роботу GPS модуля.

Android SDK можна поділити на декілька груп: один із них містить дані з модуля для розробки мобільних додатків, а саме для мобільної платформи Android, ці модулі також містять певні робочі засоби та компоненти для специфічних пристроїв, наприклад, планшет Samsung Galaxy. Інша група модулів містить в собі всі інші модулі, наприклад код, документацію, API, служби Google.

Модуль платформи SDK зазвичай розпаковується до каталогу ~ / android / adt-bundle- <OS-PLATFORM > платформа / платформа-NNN, де NNN - номер версії платформи API (номер). Для кожного основного видання платформи була

опублікована нова версія API, наприклад, для Android 2.2, номер версії API 8, для Android 2.3.1 - 9, для Android 2.3.3-10, для Android 4.2.2 - 17 і так далі. Модуль містить файли, необхідні для запуску цієї платформи в емуляторі платформи Android. Але в цьому модулі Google не встановлюються, наприклад, для Google Maps. Модулі підтримки Google API позначені окремо і зазвичай називаються Google API від Google Inc. Загалом, всі модулі, розміщені в платформах ~ / android / adt-bundle- <OS-PLATFORM> /, мають приблизно однакову структуру містять файли, з яких створюється зображення віртуального пристрою AVD.

У Android SDK є засоби підключення до пристрою з Android OS, причому вони працюють абсолютно однаково як з реальними, так і з віртуальними. Один з них називається Android Debug Bridge — це утиліта командного рядка, називається adb, лежить в каталозі ~ / android / adt-bundle- <OS-PLATFORM> / platform-tools і дозволяє виконувати налагоджувальні роботи на підключеному пристрої. Всі доступні опції програми adb можна подивитися командою adb help, вона покаже довгий список всіляких опцій з досить докладним описом кожної.

Програми для Android упаковані у форматі .apk та зберігаються в папці / data / app на ОС Android (папка доступна лише для користувача root з міркувань безпеки). Пакет APK містить файли .dex (скомпільовані файли кодів байтів, які називаються виконуваними Dalvik), файли ресурсів і т.д.

Код, написаний на C / C ++, може бути скомпільований у ARM, або x86 рідний код (або їх 64-бітові варіанти) за допомогою Android Native Development Kit (NDK). NDK використовує компілятор Clang для компіляції C / C ++. GCC включали до NDK r17, але видаляли в r18 в 2018 році.

4.2 Структура та основні компоненти Android SDK

Структура (рисунок 4.1) створеного проекту Android досить велика, вона включає у себе як і самі програмні модулі, наприклад, файли з класами, функціями та додатками, так і системні файли, наприклад маніфест додатку, файли системи автоматичної сборки проекту. Також до самого проекту входить папка з ресурсами мобільного додатка:

- Файли розмітки екранів;
- Файли розмітки певних компонентів та віджетів;
- Набір строкових ресурсів;
- Набір стилів;
- Набір векторних зображень;
- Набір растрових зображень;
- Набір кольорів у шістнадцятковому форматі;
- Набір примітивів;

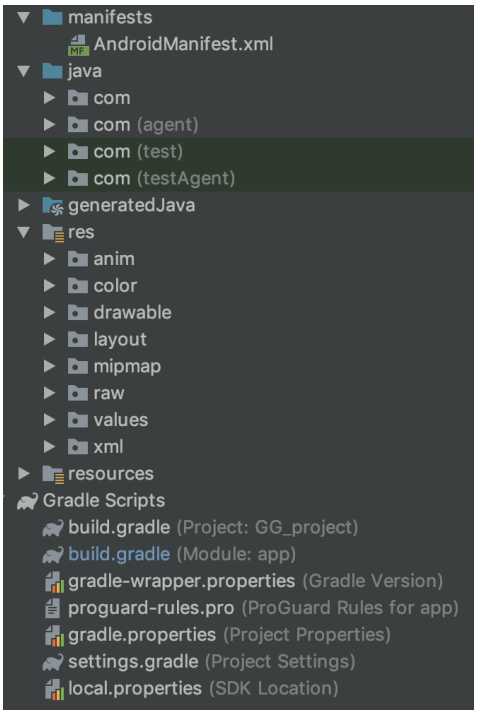

Рисунок 4.1 — Структура проекту.

4.2.1 Файл Manifest.xml

Файл Manifest має у собі головну системну інформацію про додаток. Маніфест є дуже важливим елементом проекту, він являє собою XML-файл, в якому описані властивості додатка (назва, опис), залежно та вимоги до платформи на пристрої, функціонал , який надає пакет і інші подібні речі. Для редагування маніфесту можна користуватись вбудованим в ADT редактором. В цьому файлі також необхідно явно вказати контекстні компоненти Android, які дозволи має додаток, наприклад, доступ до інтернету або доступ до галереї. В полі <application> цей файл має містити певні налаштування проекту:

- Поле Version code — номер версії програми у вигляді цілого числа, кожна наступна версія програми повинна мати більший, ніж попередні. Саме це значення використовується системою для порівняння версій;
- Поле Version name — версія додатка у вигляді довільного рядка, тут допускається практично що завгодно, система це поле використовує тільки в якості опису;
- Поля *Shared user id* и *Shared user label* — використовуються для групування різних додатків від одного виробника, додатки, у яких збігаються Shared user id і які підписані однаковим сертифікатом, можуть отримувати доступ до ресурсів один одного.

4.3 Система автоматичної збірки Gradle

Завдання (task) є основною складовою процесу будівництва (білду) Gradle. Завданням називається набором будівельних інструкцій, що запускає Gradle під час створення програми. У порівнянні з іншими білд-системами, завдання можуть бути звичайними відомими абстракціями. Gradle працює на Java Virtual Machine, використовує бібліотеки системи збірки Ant, користується інструментами управління Apache Ivy та іншими інструментами (TestNG, JUnit, SureFire, і т.п.)

4.4 Система контролю версій Git

Інструмент Git — це система контролю версій, що забезпечує версійність коду програмного продукта. Був створений Лінусом Торвальдсом для керування розробки ядра Linux. Система контролю версій - це система, що має керувати або записувати зміни у файлі або декількох файлах протягом деякого часу. Це зроблено для того щоб можна було вернутися до певної версії пізніше. Основним плюсом цієї системи є те, що існує багато сервісів, віддалених репозиторіїв, що мають можливість загружати код проекту на їх сервер, що скорочує ймовірність загубити код. Такими сервісами, наприклад, є: BitBacket, GitLab, GitHab. До того ж, ця система використовує децентралізовану систему контролю версій (рисунок 4.2). Це значить, якщо вмирає якийсь сервер, будь-який з репозиторіїв може бути відновлений.

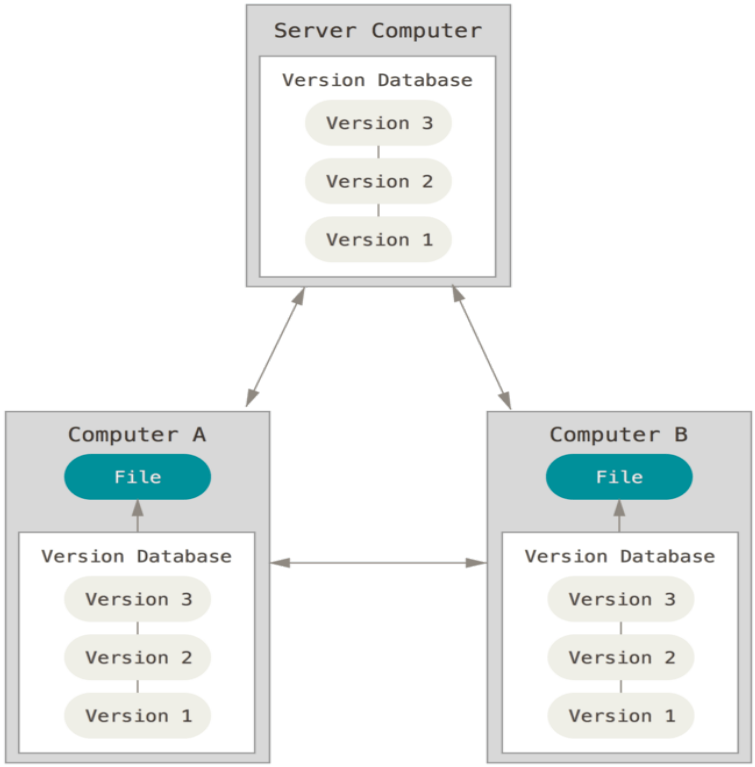

Рисунок 4.2 — Децентралізована система контролю версій

Система Git має три основні стани (рисунок 4.3):

- Збережений у коміті (commit) — це значить що файл збережений у базі даних;
- Змінений — це значить що файл збережено у базу даних, але містить певні зміни які ще відсутні;
- Індексований - це стан, який виникає тоді, коли змінений файл помічають для того, щоб його зміни були внесені у базу даних під час наступного коміту.

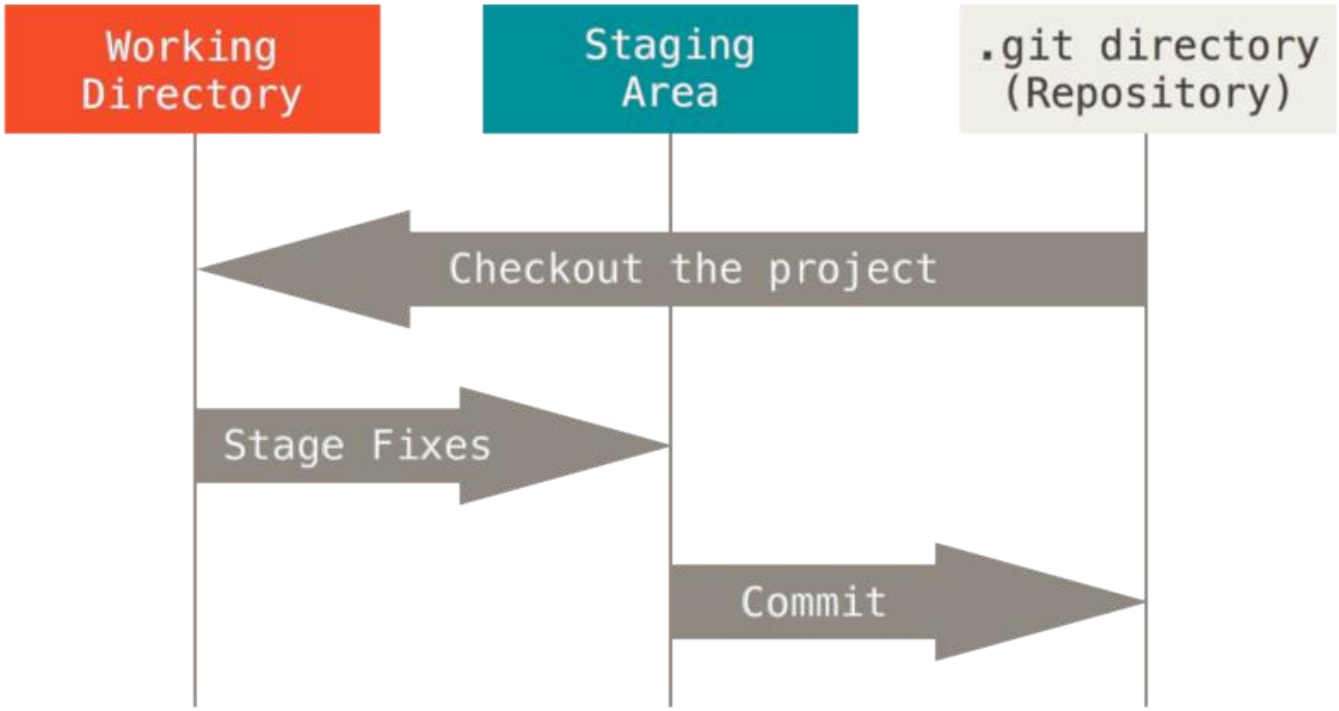

Рисунок 4.3 — Схема зміни станів.

Система Git зберігає як метадані так і базу даних об’єктів проекту. Саме ці речі копіюються при клонуванні репозиторія Git. Індекс — це файл, який зазвичай знаходиться в каталозі Git і містить інформацію про те, що буде збережено в наступній команді компіляції. Крім того, цей файл називається "Область постановки". Кожен файл цієї системи може мати два стани, або контрольований (tracked), або ні (untracked). Після того, як був зроблений клон проекта, усі файли мають контрольований статус. По мірі редагування файлів проекту, гіт їх помічає статусом змінений (modified). Якщо змінені файлі замітити, вони помічаються статусом — незміненій (unmodified). Схема трекінгу файлів прикладена нище (рисунок 4.4).

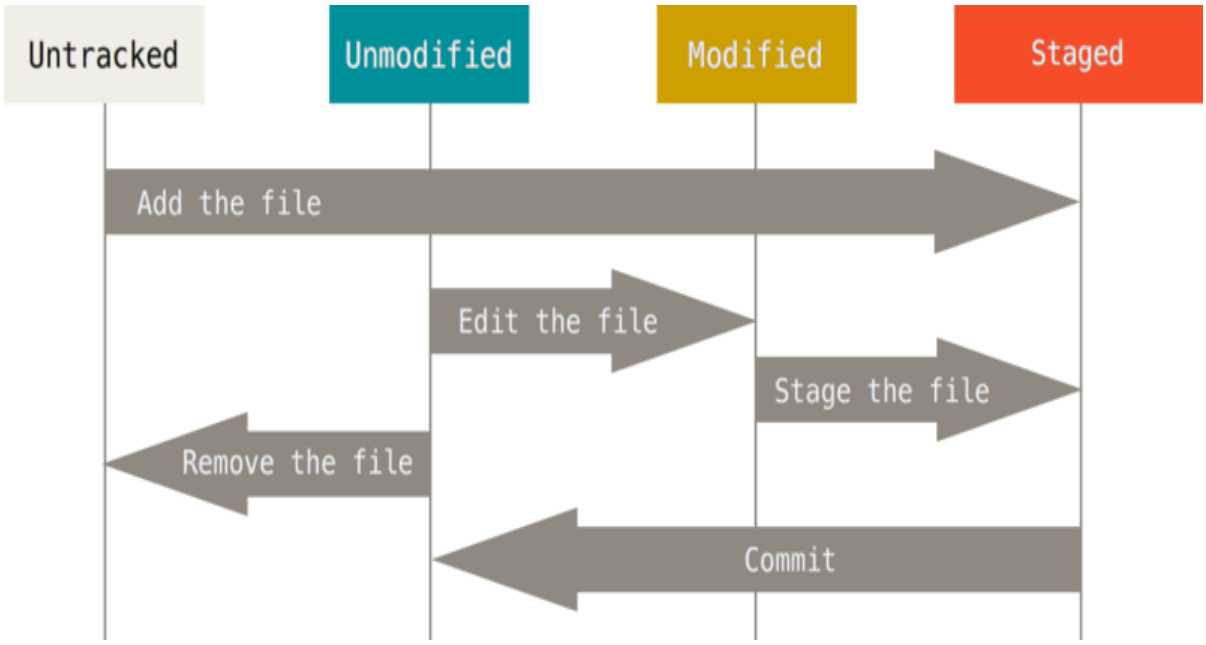

Рисунок 4.4 — Схема трекінгу файлів

Однією з основних команд є “git status” (рисунок 4.5)

```
MBP-Vladisla:maydanchik vladyslavboiko$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   app/src/main/java/com/kpi/maydanchik/data/events/EventType.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/ApiPlaygroundRepository.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/Playground.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundRepository.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundService.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/CreateActivity.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/CreatePresenter.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/CreateView.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/di/CreateComponent.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/di/CreateModule.kt
```

Рисунок 4.5 — Консольний вивід команди “git status”

Ця команда дозволяє продивитися список усіх модифікованих та неконтрольованих файлів, що знаходяться у системі Git. Наступною основною командою є “git add . ”. Ця команда дозволяє проіндексувати всі змінені та неконтрольовані файли проекту.

Найголовнішою командою являється “git commit -m <message>”. Ця команда створює коміт (рисунок 4.6) та зберігає всі відслідковані зміни в файлах проекту.

```
MBP-Vladisla:maydanchik vladyslavboiko$ git add .
MBP-Vladisla:maydanchik vladyslavboiko$ git commit -m "Your commit message"
[master dc31102] Your commit message
 63 files changed, 1577 insertions(+), 363 deletions(-)
 rewrite .idea/codeStyles/Project.xml (81%)
 create mode 100644 .idea/encodings.xml
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/events/EventType.kt
 rewrite app/src/main/java/com/kpi/maydanchik/data/events/Events.kt (78%)
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/ApiPlaygroundRepository.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/Playground.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundRepository.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundService.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/screens/create/CreateActivity.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/screens/create/CreatePresenter.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/screens/create/CreateView.kt
```

Рисунок 4.5 — Консольний вивід команди “git commit -m <message>”

4.5 Середовище розробки Android Studio

Середовище розробки Android Studio — це інтегроване середовище розробки (IDE) для роботи з платформою Android. Це середовище засноване на програмному забезпеченні Intellij Idea від розробників JetBrains. З 2017 року є офіційним середовищем для розробки Android додатків, та є рекомендованою Google. Android Studio підтримує такі мови програмування як Java, C++, Kotlin. Вбудована підтримка XML. Середовище розробки адаптоване для виконання типових завдань, які вирішуються під час розробки додатків платформи Android. У тому числі середовище включає в себе інструменти для спрощення тестування програмного забезпечення на сумісність з різними версіями платформи і засоби для проектування додатків, що працюють на пристроях з різною роздільною здатністю (планшети, смартфони, ноутбуки, годинники, окуляри і т.д.). На додаток до функцій, доступних в IntelliJ IDEA, Android Studio має кілька додаткових функцій, таких як нова уніфікована підсистема для встановлення, тестування і розгортання додатків,

заснована на інструментах Gradle Collection і підтримує використання інструментів безперервної інтеграції. Основний функціонал IDE:

- Можливість роботи з компонентами інтерфейсу користувача за допомогою функції перетягування для перегляду компонування в декількох конфігураціях екрану;
- Створення програми на основі Gradle;
- Генерація різних варіантів APK та App Bundle;
- Рефакторинг коду;
- Статистичний аналізатор коду, що дозволяє виявити проблеми які стосуються із несумісністю версій та продуктивності мобільного додатка;
- Можливість обфускувати код за допомогою ProGuard;
- Можливість підписувати APK файл власними ключами;
- Можливість користування основних шаблонів Android;
- Підтримка Android NDK;
- Вбудована підтримка певного функціоналу Google Cloud Platform;
- Вбудована підтримка Google Cloud Messaging та Firebase;
- Вбудована підтримка Kotlin та KotlinX;
- Наявна можливість розробки додатків для Android Wear, Android TV.

Однією з найголовніших функцій цього середовища є розумний редактор коду. Він дозволяє користувачеві швидше використовувати певні функції та методі, дивитися список пропозицій функцій, методів, полів від середовища (рисунок 4.6), що надзвичайно спрощує процес розробки.

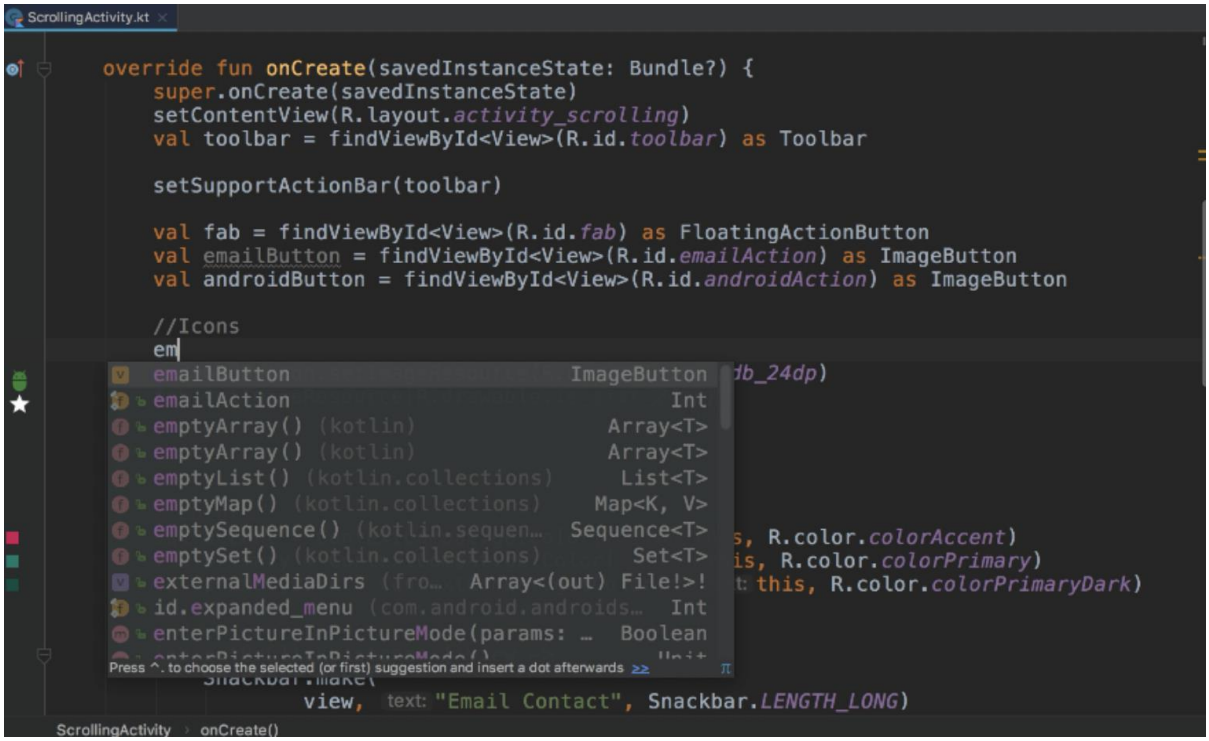

Рисунок 4.6 — Розумний редактор коду

4.6 Мови програмування

4.6.1 Аналіз мови програмування Java

Мова програмування Java — це сильно типізована мова для об’єктно-орієнтованого програмування, що була розроблена компанією Sun Microsystems, в подальшому була куплена компанією Oracle. Зараз, до OpenJDK — пакет з основними інструментами розробки, наприклад, бібліотеки, вносять свій вклад багато великих компаній таких як: Oracle, RedHead, IBM, Google, JetBrains. Додатки Java (не тільки мобільні додатки) транслюються в спеціальний байт-код, який був розроблений для віртуальної машини Java (JVM). Завдяки цієї машини та байт-коду, додатки Java можуть виконуватися на будь-якій платформі, операційній системі де встановлена JVM. На 2019 рік, ця мова програмування є найпопулярнішою у світі. Один із значних плюсів цієї мови є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, що теоретично можуть перевищувати повноваги певної програми, наприклад несанкціонований доступ до інтернету чи доступ до деяких даних, викликають

одразу преривання виконання програми. Один із мінусів це більш низька швидкість роботи, порівняно з C або C++. По деяким джерелам інформації, швидкість приблизно в два рази нижча. Але з кожним оновленням початкового коду, швидкість виконання Java програм збільшується. Найбільшим плюсом цієї мови це безпека роботи з пам'яттю, оскільки в неї нема ні вказівників, ні явних посилань на об'єкти. До того ж, швидкість розробки на цій мові набагато швидша ніж на нізькорівневих мовах програмування, таких як C або C++. Згідно з таблиці 4.1, плюсів використання цієї мови більше ніж мінусів.

Таблиця 4.1 — Порівняння плюсів і мінусів мови програмування Java

Плюси мови програмування Java	**Мінуси мови програмування Java**
Об'єктно-орієнтована мова	Відсутність нативного дизайна та якісних інструментів для дизайну розробки GUI
Високорівневий язик із простим синтаксисом	Низька швидкість роботи (в порівнянні з низькорівневими мовами)
Є стандартом для корпоративних обчислювальних систем	Платне комерційне користування
Безпечність	Багатослівний та складний код
Незалежність від платформи	Відсутність можливості писати код в не об'єктно-орієнтованому підході
Мова для розподіленого програмування і комфортної віддаленої спільної роботи	
Багатопоточність	
Стабільність та велике співтовариство	

4.6.2 Аналіз мови програмування Kotlin

Мова програмування Kotlin — це OSS-статично типізована мова, яка має підтримку Java, JavaScript та Native. Ця мова була розроблена у 2010 році компанією JetBrains, версія 1.0 була випущена у 2016 році. З 2019 року являється рекомендованою мовою для розробки платформи Android. Ця мова програмування є у відкритому доступі,

тобто “опенсорсною”, весь початковий код лежить у відкритому доступі на GitHub. Мова Kotlin являється як об’єктно-орієнтованою мовою, так і функціональною. Це значить що мова переймає властивості Java в контексті ООП, але має можливість писати у функціональному стилі, наприклад писати функції без обгортки в клас.

Kotlin може бути використаний для розробки платформ, наприклад, будь-який сервер, клієнт, веб або Android. З релізом Kotlin / Native (який на даний момент розробляється) з'являється підтримка таких платформ, як MacOS і iOS. Багато синтаксису взято із таких мов програмування: Паскаля, TypeScript, Haxe, PL / SQL, F #, Go і Scala, C ++, Java, C #, Rust і D.

Ця мова, завдяки тому, що вона перейняла найкраще з багатьох мов є набагато лаконічнішою. Програмний код на Kotlin в середньому на 40% коротший ніж код на Java. На сьогоднішній день, майже всі сучасні IDE підтримують Kotlin, серед основних можна виділити: Intellij Idea, Android Studio, Eclipse, Visual Studio Code.

Також Kotlin є типобезпечним. Завдяки null-безпеці додатки менш уразливі до NullPointerException. Також Kotlin має багато інших плюсів, в порівнянні з Java: функції вищого порядку, функції-розширення і лямбда-вирази з одержувачами^ розумне приведення типів. Це дозволяє програмісту писати виразний код і підтримувати створення DSL.

4.6.3 Перелік переваг мови Kotlin над Java

Питання полягає в тому, чи варто використовувати Kotlin замість Java чи ні. Найголовнішою відмінністю цих мов програмування є те, що в Kotlin присутня null-безпека, що дозволяє не використовувати блоки try catch. Це значить, що в Kotlin відсутні так звані Checked Exceptions, тому немає необхідності постійно перевіряти чи відловлювати виключення.

Також, Kotlin дозволяє створювати додаткові потоки як і Java, але він впроваджує новий механізм керування цими потоками, що називаються співпрограми (coroutines). Співпрограми реалізовані без використання стеку, а це значить що вони використовують меньше пам'яті ніж звичайни потоки Java.

В реальних проектах, які набувають значних розмірів, часто використовують так звані POJO класи - Plain Old Java Object. Такі класи зазвичай описують модель якоїсть сутності, їх призначення полягає в тому, що вони зберігають лише дані, та описують їх. В мові програмування Kotlin є простий спосіб створення таких об'єктів. Достатньо перец словом class написати специфікатор data. Такий клас автоматично створює геттери, сеттери, створює певні конструктори класу. До того ж перевизначае такі стандартні функції як toString(), hashCode(), equals(). Це значно спрощує розробку програмного продукта.

Найголовнішим плюсом у бік Kotlin є те, що вона має підтримку функцій вищого порядку та лямбда-функцій. Оскільки мінімальною версією Android було поставлено 5.1, це значить що в проекті буде використовуватися лише Java 1.6, в якій цей функціонал відсутній. Як статично типізований мова програмування, Kotlin використовує ряд функціональних типів для представлення функцій. Звичайні ж функції в мові Kotlin є функціями першого порядку, це значить що вони можуть зберігатися у структурах даних та у вигляді полів, також можуть передаватися в якості аргументів.

Іншим плюсом є те, що Kotlin надає можливість розробниками використовувати так звані функції розширення. Вони дозволяють розширювати певні класи не змінюючи їх зсередини.

Висновки до розділу 4

В цьому розділі було проведено детальний перегляд Android SDK, особливостей, компонентів. Було обрано нативний додаток в якості розробки, через ефективність та швидкодію. Було розглянуто систему контролю версій Git, показано необхідність застосування у реальному проекті. Показано детальну роботу с цією системою, наведено приклади роботи в проекті. Розглянуто систему автоматичної збірки проекту, її роботу в системі розробки на платформы Android. Розглянуто принцип роботи з програмним середовищем Android Studio, її переваги та мінуси. Розглянуті основні особливості мов Kotlin та Java, порівняно їх переваги та мінуси. Обгрунтовано вибір Kotlin в якості мови програмування та розробки мобільного додатка.

5 АРХІТЕКТУРА ТА БІБЛІОТЕКИ ДЛЯ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

Вдала архітектура надає шанс розробникам ніколи не проводити глобальний рефакторинг всього застосунку або програмного продукту[3]. Якщо правильно спроектувати проект, вибрати коректну архітектуру, то можна дуже спростити собі життя. Найзначніші зміні - це переробка певних модулів коду без зміни клієнтського. При неправильно вибраній архітектурі, с кожною строкою коду збільшується складність у підтримці, розширені та тестуванні цього проекту.

5.1 Архітектура MVP, Clean Architecture

Архітектура MVP — розшифровується як Model View Presenter. Це одна найпопулярніша та із основних архітектур для розробки Android. Цей шаблон проектування надає можливість програмісту розподілити відповідальності при розробці користувацького інтерфейсу.

Основним компонентом в системі Android є активність (Activity), вона має у собі всю логіку, відповідальна за дію користувачів, бізнес логіку, представлення, роботу з базами даних та інтернетом. Якщо не розподілити ці відповідальності на класи то можуть бути складнощі з розширенням та підтримкою програми. В цій архітектурі є 3 основних сутності:

- Сутність View — це клас, що відповідальний за всю взаємодію з користувачем.
- Сутність Presenter — це клас, що відповідальний за основну бізнес логіку додатку.

- Сутність Model — це класи, які відповідають за всі дані мобільного додатку, їх маршрутизацію, процес збереження та інше.

Для того щоб зрозуміти за що конкретно відповідають ці сутності на платформі Android можна подивитися таблицю 5.1.

Таблиця 5.1 — Відповідальності сутностей в MVP

Model	**View**	**Presenter**
Відповідає за маршрутизація даних	Відповідає за створення сутності Presenter та механізм його приєднання або від'єднання;	Відповідає за завантаження моделей
Відповідає за взаємодію з базою даних	Відповідає за оповіщення Presenter про важливі життєві події додатку	Відповідає за збереження посилання моделей
Відповідає за взаємодію з різними джерелами даних такими як інтернет	Відповідає за оповіщення Presenter про події від користувача	Відповідає за форматування та маппінг, того що повинно бути відображено на View
Відповідає за моделі	Відображає дані, що були оброблені та повернуті Presenter	Відповідає за команди до View, що вона повинна показати, зробити і т.п.
	Відповідає за розміщення View - елементів та інших віджетів	Взаємодія з репозиторіями
	Відповідає на переходи до інших екранів мобільного додатка	Відповідає за визначення певних дій після того як отримані певні події від View
	Відповідає за анімації	

Кожна сутність взаємодіє лінійно (рисунок 5.2), наприклад, View не має доступ до Model, лише до Presenter. Так само і Model не має доступу до View[2].

Сутність Presenter взаємодіє як з Model так і з View, та виступає певним мостом між цими сутностями. Схема взаємодії представлена нище (рисунок 5.1).

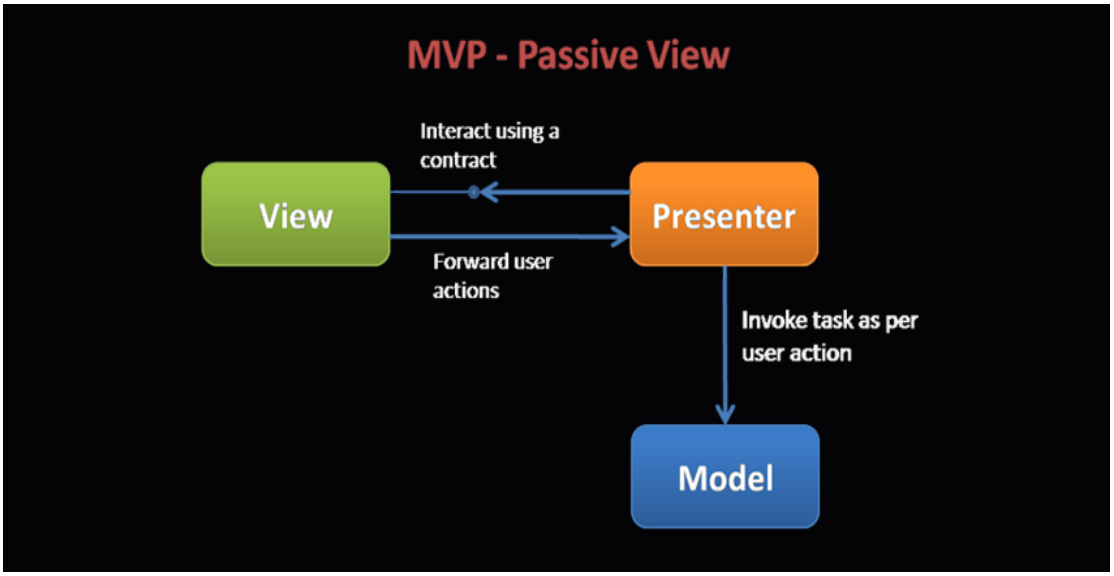

Рисунок 5.1 — Схема взаємодії сутностей MVP

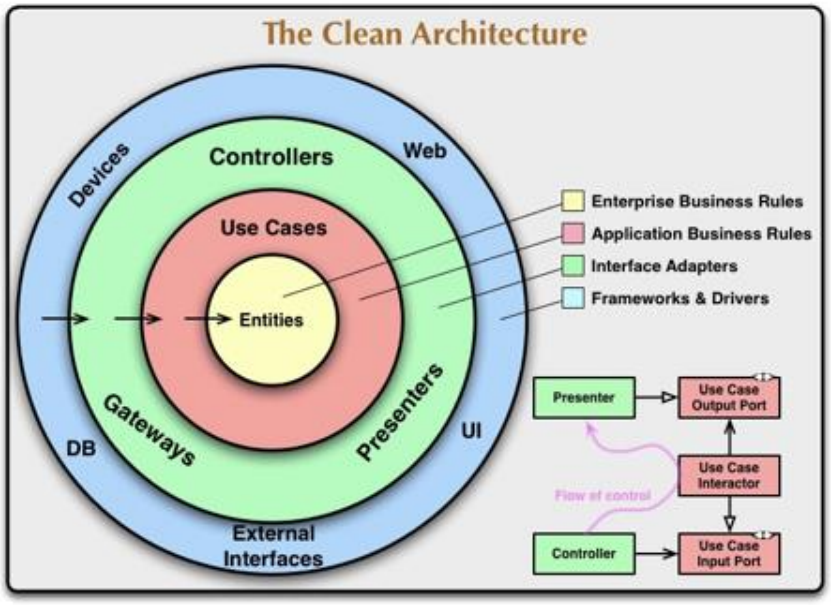

Рисунок 5.2 — Схема потоку даних в MVP

Як було вище сказано, такий архітектурний підхід спрощує тестування мобільного додатку, оскільки більшість коду, яку необхідно протестувати буде знаходитись в Presenter. До того ж, ця сутність незалежна, тому можна тестувати лише цей модуль.

5.2 Фреймворк Dagger 2

Dagger 2 — це повністю статичний фреймворк для впровадження залежностей.

Цей фреймворк використовують для реалізації шаблону IoC (Inversion of Control), що дозволяє зменшити зчепленність в проекті. Це набагато полегшує розширення системи та його тестування. Одним із реалізацій цього шаблону є впровадження залежностей. Є різні типи впровадження залежностей:

- Впровадження через конструктор;
- Впровадження через метод класу;
- Впровадження через інтерфейс впровадження;

Фреймворк має можливість використовувати усі ці типи впровадження. Він працює на основі кодогенерації. Також Dagger створює екземпляри класів додатків і задовольняє їх залежностям. Він використовує анотацію javax.inject.Inject, щоб визначити, які конструктори та поля цікавлять. Основні анотації фреймворку:

- Анотація Inject
- Анотація Module
- Анотація Component
- Анотація Provide
- Анотація Singletone

Для того щоб задовольнити залежність Inject, потрібно її створити та надати. Усі залежності створюються в спеціальній сутності Module — звичайний клас, в якому містяться методи що помічені анотаціями Provide. Ці методи, зазвичай, створюють, або використовують готові залежності для задоволення інших залежностей що помічені анотацією Inject.

Оскільки модулів може бути багато, і для створення однієї залежності може потребуватися інша, було впроваджено спеціальну сутність Component. Це

інтерфейс, який містить список Module, та список классів, в які можна впроваджувати ці залежності (рисунок 5.3).

```
@Singleton
@Component(modules = [ApplicationModule::class])
interface ApplicationComponent {

    fun inject(app: MaydanchikApp)

    fun serviceProvider(): ServiceProvider

    fun preferencesRepository(): PreferencesRepository

    fun eventRepository(): EventsRepository

    fun workers(): Workers

    fun userRepository(): UserRepository
}
```

Рисунок 5.3 — Компонент Dagger 2

Кожен компонент створює граф залежностей, це значить що він пов’язує між собою модулі та надає тим самим можливість взаємодіяти. Наприклад, якщо одному з модулів необхідна залежність з іншого модуля, для того щоб створити власну, то компонент надає доступ до цієї залежності.

До того ж, цей фреймворк надає можливість створення Sigleton — породжуючий шаблон програмування, що гарантує, що для певного класу буде створений тільки один об’єкт, а також буде наданий глобальний до нього доступ. Фреймворк Dagger реалізував цей шаблон[11], його можна впровадили використання всього однієї анотації @Singleton. Також він надає можливість створити власні анотації, наприклад для локальних Singleton, це може бути корисно, коли необхідно задати певні межі життя об’єкту.

5.3 Бібліотека Retrofit

Однією із основних бібліотек для роботи з інтернетом в Android є бібліотека Retrofit. На 2019 рік вона вважається певним стандартом для розробки програмного забезпечення на Android. Однією із причин популярності є те, що вона відмінно підтримує REST API, та досить швидко працює із HTTP. Також ця бібліотека відмінно взаємодіє із RxJava, мова про яку піде далі.

Бібліотека дозволяє робити усі типи HTTP запитів, включаючи Mutlipart запити, що можуть знадобитися для передачі певних файлів, наприклад картинок. Ця бібліотека використовує анотації у якості позначення типів інтернет-запитів:

- Анотація GET;
- Анотація POST;
- Анотація PUT.

Також, для того щоб передати певні параметри використовуються анотації типів цих параметрів:

- Анотація BODY;
- Анотація PATH;
- Анотація HEADER;

Для того щоб користуватися цією бібліотекою, потрібно створити інтерфейс, де будуть описані усі запити, які дані вони приймають та які дані вони віддають. Для того, щоб користуватися сервісом, необхідно створити посилання на клас Retrofit. Для цього використовують Retrofit.Builder(), цей об’єкт має можливість налаштовувати HTTP клієнт за бажанням користувача. Далі, необхідно використати це посилання, щоб використати метод retrofit.create(serviceClass). Його єдиним параметром є serviceClass, який є інтерфейсом який був описаний вище. Після цього, Retrofit поверне створений ним об’єкт цього інтерфейсу, що надасть можливість відправляти посилання в інтернет.

5.4 Бібліотека RxJava

Бібліотека RxJava — це Java реалізація на Reactive Extensions. Вона надає можливість впровадити у проект реактивне програмування. Реактивне програмування - це парадигма програмування, що орієнтована на потоки даних та їх розповсюдження та розповсюдження змін. Основна робота полягає в роботі з асинхронними потоками даних.

Ця бібліотека, по своїй суті, реалізує шаблон програмування “Спостерігач”. Вона надає методи для підтримки послідовностей даних та певних подій, додає велику кількість операторів що надають можливість декларативно складати та описувати послідовності дій, при цьому відводячи занепокоєння про такі речі як синхронізація, безпека потоків, безпека виводу та вводу, низькорівневі потоки.

Бібліотека Rx базується на двох фундаментальних типах, в той час, як деякі інші розширюють їх функціональність. Цими базовими типами є Observable і Observer. Тип Observable представляє собою деяку сутність що має можливість змінюватися та надавати інформацію про певні зміні своїм підписникам - Observer. Однією з особливостей цієї бібліотеки є різноманітність підписників з різними термінальними станами:

- Тип Observable;
- Тип Single;
- Тип Flowable;
- Тип Completable;
- Тип Maybe.

Кожен тип має різне призначення. Якщо програмісту необхідно отримати лише одиничну результат якогось запиту, то необхідно використати Single. Він має два стани: OnSuccess, OnError. Перший стан виконується коли приходить результат, другий, коли відбувається якась помилка.

Якщо результат не важливий, але важливо знати що дія була виконана, необхідно використати підписника Completable. Він має такі ж два стани, як і у Single, але нічого не повертає.

Тип Maybe, виходячи з його назви може або повертати результат або ні, причому в обох випадках він успішно буде виконаний.

Найскладнішими типами є Observable та Flowable — це типи що мають 3 термінальних стани: OnNext, OnSuccess, OnError. Останні два мають такі ж властивості що й у інших типів. Стан OnNext викликається тоді, коли до підписника необхідно відправити дані, але вони не останні, тому він не відпрацьовує до кінця. Основна різниця між цими двома класами — Backpressure Strategy. Термін Backpressure — це подія, коли Observable приймає набагато більше інформації, ніж його підписники можуть опрацювати, через це виникають проблеми з пам’яттю. Тип Flowable має можливість вирішувати такі проблеми через Backpressure Strategy.

5.5 Використання Google Map Api

За допомогою Google Map Api в мобільному додатку можна використовувати карти, які базуються на Google Map. Це API у автоматичному режимі оброблює доступи до інформації, що зв’язана з картами, займається завантаженням інформації, наприклад назв вулиць, міст, багатьох місць загального користування, обробкою жестів на карті. Використовуючи це API можна додавати маркери на карті, полігони, що перекривають та позначають виділену територію. Також можна впровадити кластеризацію, якщо маркерів на карті забагато. Дозволено додавати різноманітну графіку до фрагменту карти:

- Значки, прив'язані до певних позицій на карті (Маркери).
- Набори відрізків ліній (поліліній).
- Вкладені сегменти (багатокутники).
- Графічна бітова графіка прив'язана до певних позицій на карті (наземні накладання).
- Набори зображень, які відображаються у верхній частині основних плиток карти (Tile Overlays).

Для того щоб використовувати це API, необхідно отримати спеціальний ключ через сервіс Google Cloud Platform, він використовується майже з усіма сервісами зв’язаними з геолокацією та картами.

5.6 Використання Google Static Map

Завдяки використанню Google Static Map можна значно зекономити ресурсі при необхідності показати карту з якою неможливо взаємодіяти. Це можуть бути невеликі фрагменти з локацією певного об’єкта, який додається до опису. Для використання цього API також необхідно отримати спеціальний ключ, який буде використовуватися у параметра запиту.

Головним плюсом є те, що для того щоб отримати карту, необхідно зробити запит в інтернет з певними параметрами. Цей запит (рисунок 5.4) в результаті поверне картинку, яку й використовують для відображення карти.

```
https://maps.googleapis.com/maps/api/staticmap?center=Brooklyn+Bridge,New+York,NY&zoom=13&size=600x30
&markers=color:blue%7Clabel:S%7C40.702147,-74.015794&markers=color:green%7Clabel:G%7C40.711614,-74.01
&markers=color:red%7Clabel:C%7C40.718217,-73.998284
&key=YOUR_API_KEY
```

Рисунок 5.4 — Приклад запиту до Google Static Map Api

Одним із плюсів є те, що ця карта надзвичайно гнучко настроюється. Велика кількість параметрів надає можливість не тільки відключити або включити певні маркери, а й змінити колір, наприклад, будівель та доріг. Завдяки цієї гнучкості можна додавати власні маркери та навіть використовувати для цього власні картинки.

Одним із обов’язкових параметрів є key. В цьому параметрі має бути присутній власний API ключ. Наступними головними параметрами є координати центру карти, вони описуються словами lat та lnt. Де lat - це широта, а lng - це довгота. Для того, щоб налаштувати стилі об’єктів на карті, використовуються три основних параметри:

- Параметр Feature;
- Параметр Element;
- Набір стильових правил.

Параметр Feature відповідає за вказівку функцій, що потрібно вибрати для модифікації стилю, це можуть бути дороги, будівлі, визначні місця. Якщо цей параметр не буде вказано, то на карті будуть усі елементи, за які він відповідає.

Параметр Element відповідає характеристики функцій, це можуть бути геометрія або мітки. Це не обов'язковий елемент, якщо він буде відсутній у запиті, то стиль застосовується до всіх елементів зазначеної функції.

Набір правил стилю застосовуються в тому порядку, в якому вони відображаються в декларації стилів (рисунок 5.5).

```
style=feature:myFeatureArgument|element:myElementArgument|myRule1
```

Рисунок 5.5 — Приклад застосування стилів

Також було створено власний запит з використанням стилів та маркеру на карті (рисунок 5.6).

```
fun getMapUrl(location: LatLng): String {
    return "https://maps.googleapis.com/maps/api/staticmap?center=${location.latitude},${location.longitude}" +
            "&zoom=16&size=800x600&markers" +
            "=icon:https://s3.eu-west-2.amazonaws.com/rptestbucket1/photo_2018-08-29_11-54-59.jpg" +
            "|${location.latitude},${location.longitude}&style=feature:landscape" +
            "|color:0xffffff&style=feature:poi.park" +
            "|element:geometry|color:0xebf5eb&style=feature:poi" +
            "|element:geometry|color:0xebf5eb&style=feature:landscape.man_made" +
            "|element:geometry.fill|visibility:off&style=feature:water" +
            "|color:0xc9c9c9&style=feature:poi.park|color:0xebf5eb&style=feature:road.highway" +
            "|color:0xF3F8FF&style=feature:labels.icon|visibility:off&style=feature:transit.line" +
            "|color:0xe5e5e5&style=feature:transit.station|color:0xeeeeee&style=feature:geometry" +
            "|color:0xeff2f4&key=AIakj12k1jd91lkLAK0asdmj"
}
```

Рисунок 5.6 — Приклад запиту з власним стилем

Висновки до розділу 5

В цьому розділі було обрано архітектуру для мобільного доданку – MVP. Були наведені основні плюси цієї архітектури, її ідею, та детально описано взаємодію компонентів системи та основні сутності. В якості фреймворку для впровадження Dependency Injection вибрано Dagger 2, що значно полегшує впровадження залежностей. В якості http бібліотеки вибрано Retrofit через її простоту у реалізації та опису. Розказано про основні типи запитів та їх реалізація. З'ясовано доцільність використання Google Static Map Api та Google Map Api.

6 ПРОГРАМНА РЕАЛІЗАЦІЯ ANDROID-ДОДАТКУ ІНТЕРНЕТ-СЕРВІСУ

Однією з основних цілей дипломної роботи була реалізація мобільного додатку, призначеного для вирішення проблем з бронюванням та відвідуванням спортивних майданчиків. Програмний продукт був виконаний на платформі Android у нативному середовищі.

6.1 Загальний опис роботи активностей мобільного додатку

При першому запуску програми, відкривається головний екран MainActivity (рисунок 6.2), який був прописаний в маніфесті проекта як стартовий екран. Цей екран, до того як відбулося промальовування головних компонентів екрану — віджетів та View, робить запит до Shared Preferences для того, щоб перевірити чи авторизований користувач, чи ні.

Shared Preferences — це, по своїй суті, файл із певними записами, які зберігаються на мобільному пристрої не тільки під час роботи додатку, а й після вимкнення програми. В цьому файлі прийнято зберігати деякі системні налаштунки, що вибрав користувач та збереження токену авторизації. Саме цей токен авторизації перевіряє на наявність MainActivity.

Якщо токен є у наявності, то головний екран залишається та промальовуються основні віджети екрану, якщо токен відсутній то показується екран авторизації LoginActivity (рисунок 6.1), що надає можливість користувачеві зареєструватися або увійди.

6.1.1 Екран авторизації

Екран надає можливість користувачу виконувати реєстрацію та вхід до програми.

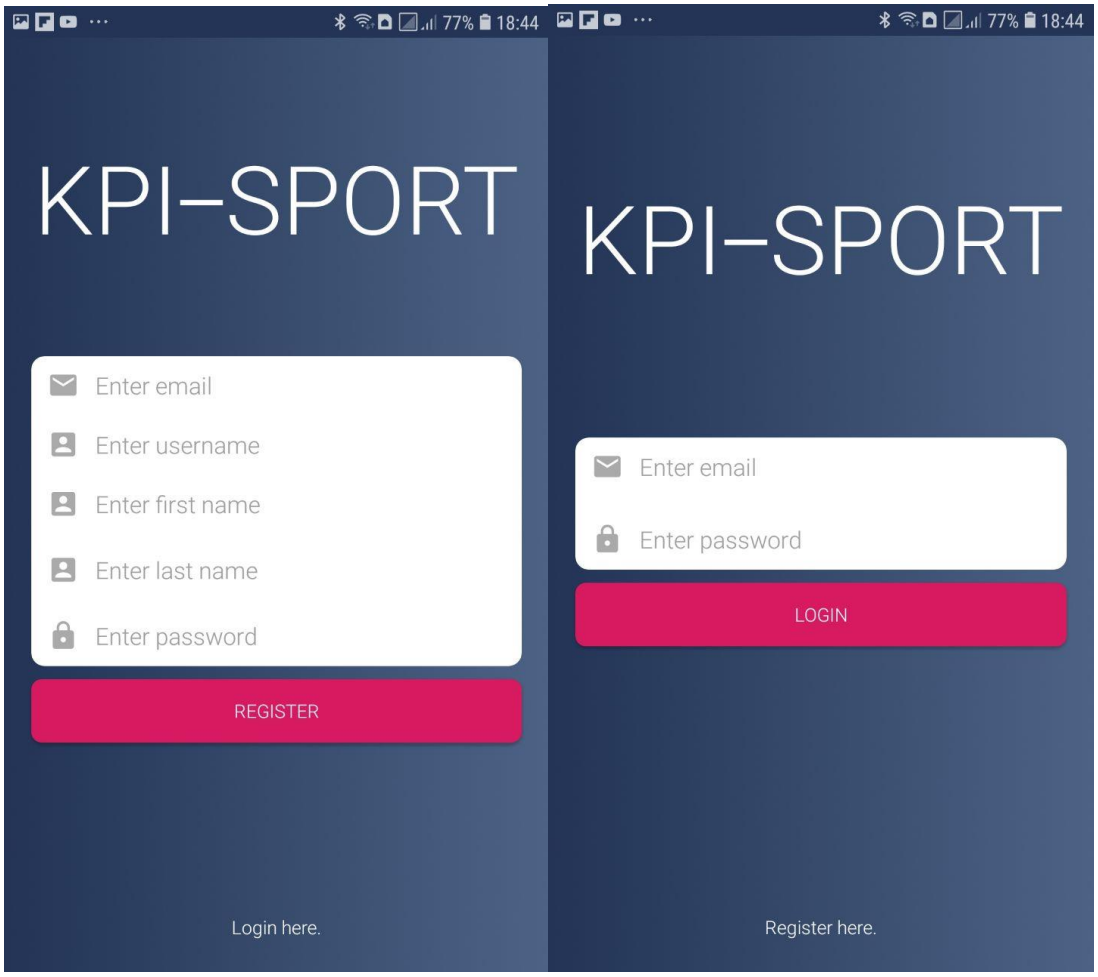

Рисунок 6.1 — Екран авторизації

При натисканні на кнопку "Register here\Login here", що знаходиться знизу екрана, починають анімовано з’являтися або зникати додаткові поля, що використовуються для реєстрації ко

ристувача. Також існує перевірка на коректність введених даних, якщо ж одне із полей буде заповнене невірно, то воно покаже сповіщення. Були взяті такі правила для перевірки введених даних:

- Поле e-mail повинне відповідати усім стандартам електронної пошти: має мати символ “@”, мінімум один символ до собаки, після собаки має йти валідне доменне ім’я
- Поле password повинне мати не меньше шести символів.
- Поля username, lastname, firstname повинні мати не меньше двох символів.

Після натискання на кнопку “Login” або “Register” відбувається запит на сервер, якщо всі поля валідні. Під час всіх довгих операцій, такі як запит в інтернет показується віджет “ProgressBar”, що означає завантаження.

6.1.2 Головний екран

Головний екран мобільного додатку має список усіх актуальних спортивних подій на всіх спортивних майданчика. Кожен елемент списку може натискатися, через користувачеві буде показаний екран події. На головному екрані також є кнопка оновити, що робить запит на сервер та оновлює весь список подій.

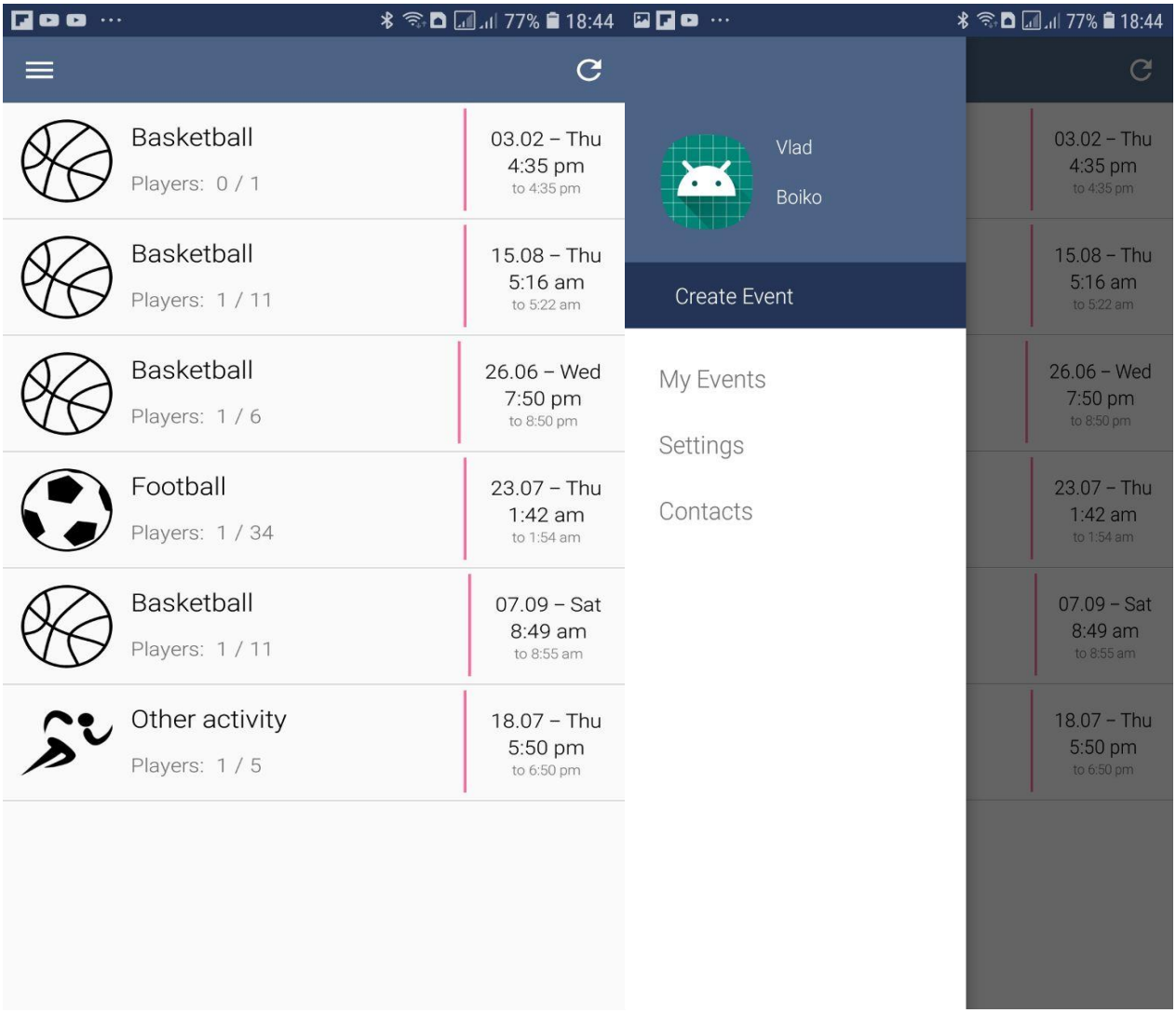

Рисунок 6.2 — Головний екран

6.1.3 Екран налаштувань

Екран налаштувань (рисунок 6.3) надає можливість користувачу змінити певну інформацію у своєму профілі. Він має 3 змінні поля, що перевіряються перед запитом на сервер для зміни інформації. Правила перевірки аналогічні до екрану авторизації.

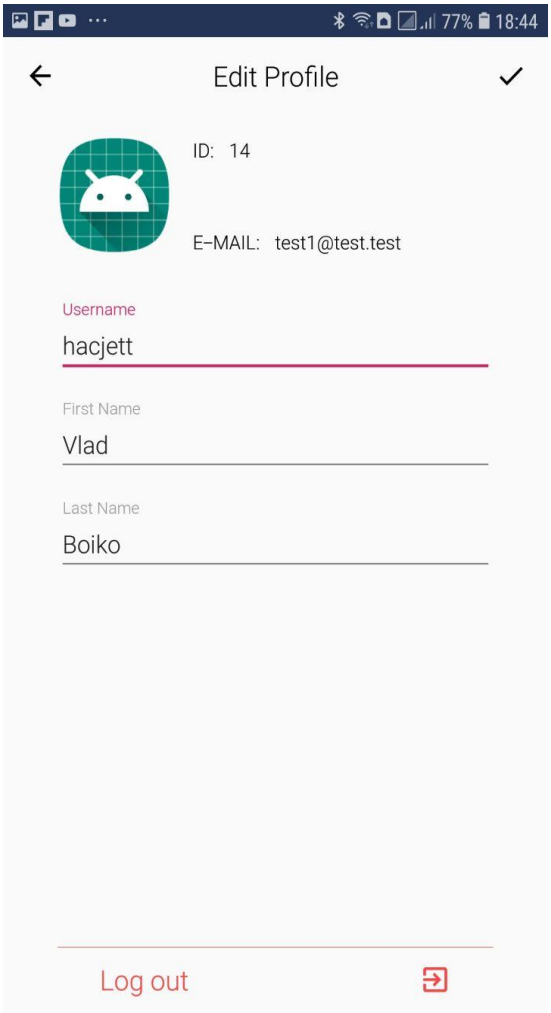

Рисунок 6.3 — Екран налаштувань

Знизу екрану знаходиться кнопка виходу із мобільного додатка. Під час виходу відбувається скидання всіх налаштувань у Shared Preferences, включаючи видалення токена реєстрації.

6.1.4 Екран спортивної події

Екран спортивної події (рисунок 6.4) містить у собі повну інформацію про спортивну подію: час початку та закінчення, назву, описання, назву та місцезнаходження спортивного майданчика, кількість зареєстрованих гравців та максимальну кількість гравців. Усі поля незмінні.

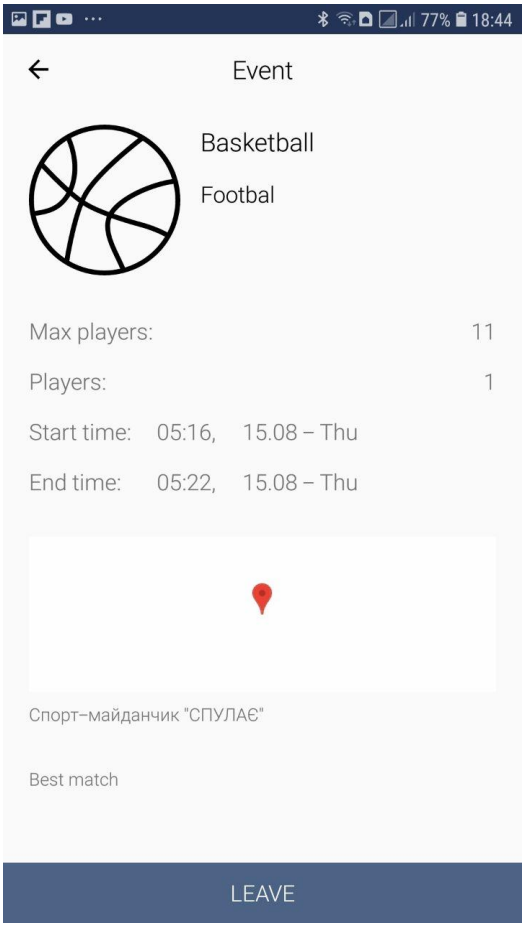

Рисунок 6.4 — Екран спортивної події

На екрані присутній віджет “ImageView”, що дозволяє відображати певні зображення. Цей екран доступається через Google Static Map та ініціалізує віджет.

До того ж, знизу екрана знаходиться кнопка, що дозволяє користувачу або під’єднатися до спортивної події, якщо він ще не під’єднаний, або від’єднатися. При кожному натисканні активність робить запит на сервер.

6.1.5 Екрани створення події

Екранів створення події два (рисунок 6.5), та вони дозволяють якісно та детально налаштувати власну подію.

Перший екран надає можливість вибрати спортивний майданчик, подивитися де він знаходиться завдяки інтерактивної карти. В ньому знаходиться випадаючий список, в якому є всі спортивні майданчики що зареєстровані в системі.

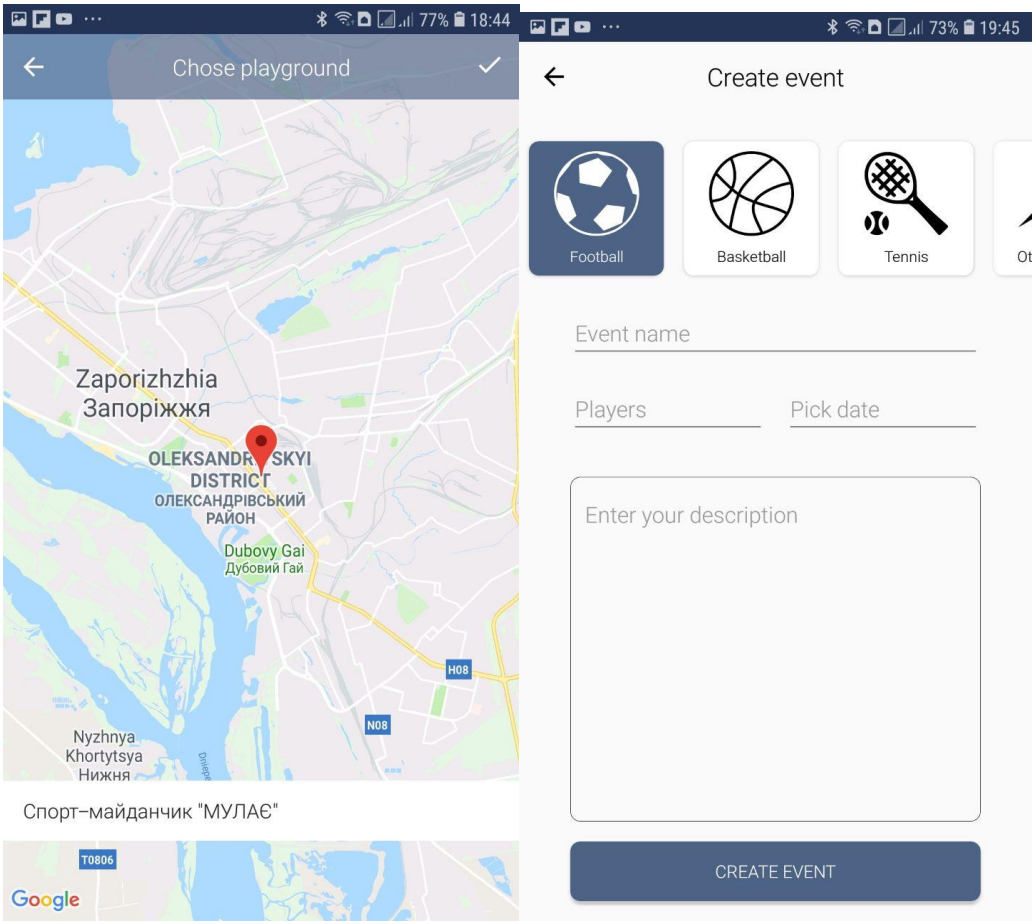

Рисунок 6.5 — Екрани створення події

Другий екран дозволяє ввести деталі щодо спортивної події. Користувач може обрати тип спортивної події, наприклад, футбол завдяки спеціальному віджету у вигляді списку кнопок з іконками. Також, користувач може ввести назву спортивної події, детальний опис події, кількість гравців та час проведення завдяки спеціальних полям вводу. При натисканні на поле “Pick date” з’являється спеціальне діалогове вікно вибору часу та дати (рисунок 6.6).

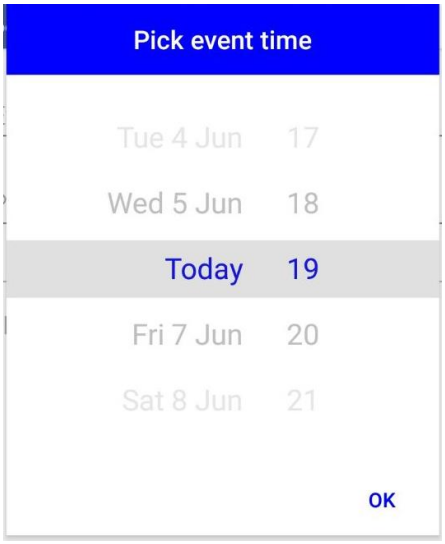

Рисунок 6.6 — Віджет вибору часу та дати

6.1.1 Екран власних подій

Екран власних подій (рисунок 6.7) надає доступ користувачу до власних подій, до яких він приєднався або які він створив. Екран має список аналогічний списку на головному екрані, він також має елементи що натискаються та ведуть по сторінки події. До того ж, зверху екрана є дві вкладки що дозволяють користувачеві змінити вивід власних подій або на створені, або на приєднані.

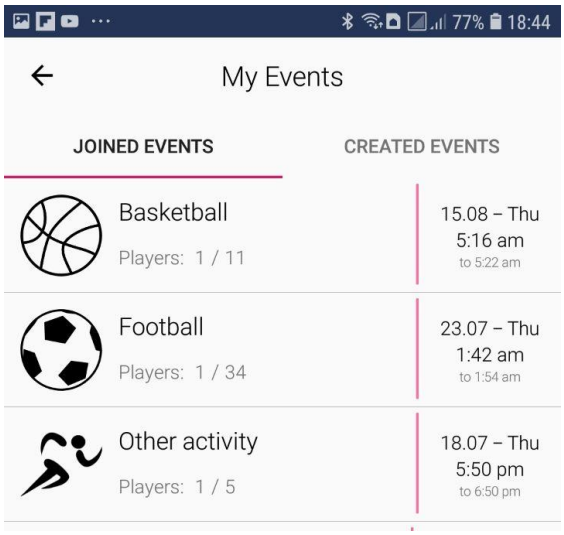

Рисунок 6.7 — Екран власних подій

Висновки до розділу 6

У розділі наведено детальний опис створення взаємодії між двома додатками. Технологія заздалегідь створеної функції має переваги щодо економії часу на виконання програми. Тому цю технологію доцільно застосувати в комплексі моделювання гідроакустичних процесів, що забезпечить більш швидке одержання результату.

ВИСНОВКИ

Проаналізовано ринок користувачів, які використовують даний тип додатків. технології створення мобільних додатків. Поставлене конкретне технічне завдання та вимоги щодо роботи мобільного додатка та його якості. Досліджені типи мобільних доданків, їх властивості та переваги. Досліджені програмні мови для розробки, визначено їх плюси, мінуси, та зроблено висновки щодо використання. Також досліджено конкурентів та аналогів даного додатка.

Було досліджено інструменти на технології для розробки мобільних додатків.

Було спроектовано архітектуру мобільного додатку, механізм спілкування програми з сервером. Реалізовано та спроектовано зручний інтерфейс системи бронювання для мобільної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гобрик, Д. А. Человеко-машинный интерфейс / Д. А. Гобрик; науч. рук. И. И. Гутич // Материалы 70-й студенческой научно-технической конференции : [тезисы докладов студентов БНТУ] / ред. колл.: Е. Е. Трофименко [и др.]. — Минск: БНТУ, 2015. — С. 45-46.
2. Проектування інформаційних систем: Посібник / За ред. В.С. Пономаренка. — К.: Академія, 2002. — 450 с.
3. Буч Г. Об’єктно-орієнтоване проектування з прикладами застосування / Г. Буч. — К.: Академія, 2002. — 723 с.
4. Вуль В.А. Загальна характеристика електронних видань [Електронний ресурс]: навч. посіб. / Вуль В.А. – Електрон. Видання. – Моска, 2011. Режим доступу: http://www.hi-edu.ru/e-books/xbook119/01/part-002.htm
5. Пантилейкин Н. В. Мобільні застосунки та їх види // Научно-методический электронный журнал «Концепт». – 2016. – 776–780с. – [Електронний ресурс]. – Режим доступу : http://e-koncept.ru/2016/46956.htm.
6. Які види мобільних додатків бувають і яке підходить у Вашому випадку. 2016. [Електронний ресурс]. – Режим доступу : https://eradv.ru.
7. Сосніна Олена. Інтерактивна книга: як мобільні пристрої змінюють обличчя книги. 2015. [Електронний ресурс]. – Режим доступу : http://pro-books.ru.
8. Мирошниченко Денис. Огляд ринку мобільних застосунків. 2017. [Електронний ресурс]. – Режим доступу : https://www.openbusiness.ru.
9. Овечкин Олег. Зростання ринку мобільних застосунків. 2016. [Електронний ресурс]. – Режим доступу : https://rb.ru.
10. Ларман, Крэг. Застосування UML 2.0 та шаблонів проектування / Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. — [3-е вид.] 2006. — 736с.

11. Паттерн Singleton (одиночка,синглет). 2012. [Електронний ресурс]. – Режим доступу : http://cpp-reference.ru.

ДОДАТОК А

Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б 12-1	maydanchik.apk	Основний компонент додатку взаємодії MATLAB з C#

ДОДАТОК Б

Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б

Аркушів 3

Київ 2019

Текст мобільного додатку

```
package com.kpi.maydanchik.screens.main

import com.kpi.maydanchik.core.base.BasePresenter
import com.kpi.maydanchik.data.events.Event
import com.kpi.maydanchik.screens.main.domain.MainInteractor

class MainPresenter(private val interactor: MainInteractor) : BasePresenter<MainView>(interactor) {

  override fun onAttach(view: MainView) {
    super.onAttach(view)
    getEvents()
  }

  private fun getEvents() {
    mvpView?.showLoading()
    interactor.getEvents(::onGetEvent, ::onError)
  }

  fun refresh() {
    mvpView?.showLoading()
    interactor.refresh({ mvpView?.hideLoading() }, ::onError)
  }

  private fun onGetEvent(events: List<Event>) {
    mvpView?.hideLoading()
    mvpView?.onGetEvents(events)
  }
}
package com.kpi.maydanchik.screens.main

import android.content.Intent
import android.os.Bundle
import androidx.core.view.GravityCompat
import androidx.recyclerview.widget.LinearLayoutManager
import com.kpi.maydanchik.R
import com.kpi.maydanchik.core.base.BaseActivity
import com.kpi.maydanchik.data.events.Event
import com.kpi.maydanchik.screens.event.EventActivity
import com.kpi.maydanchik.screens.main.di.DaggerMainActivityComponent
import com.kpi.maydanchik.screens.main.di.MainActivityModule
import com.kpi.maydanchik.screens.main.ui.EventsAdapter
import kotlinx.android.synthetic.main.activity_main.*
import javax.inject.Inject

class MainActivity : BaseActivity(), MainView {

  val component by lazy {
    DaggerMainActivityComponent.builder()
        .applicationComponent(getComponent())
        .mainActivityModule(MainActivityModule())
        .build()
  }

  @Inject
  lateinit var presenter: MainPresenter

  private lateinit var adapter: EventsAdapter

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    component.inject(this)
    presenter.onAttach(this)
    initViews()
  }

  private fun initViews() {
    filterTv.setOnClickListener { }
    refreshTv.setOnClickListener { presenter.refresh() }
    menuIv.setOnClickListener { drawer.openDrawer(GravityCompat.START) }
    initEventList()
  }

  private fun initEventList() {
    adapter = EventsAdapter(::showEvent)
```

Текст мобільного додатку

```
            eventsList.layoutManager = LinearLayoutManager(this)
        eventsList.adapter = adapter
    }

    private fun showEvent(id: Int) {
        val intent = Intent(this, EventActivity::class.java)
        intent.putExtra(EventActivity.INTENT_EVENT_ID, id)
        startActivity(intent)
    }

    override fun onGetEvents(events: List<Event>) {
        adapter.events = events
    }
}
package com.kpi.maydanchik.screens.main.di

import com.kpi.maydanchik.core.network.ServiceProvider
import com.kpi.maydanchik.data.preferences.PreferencesRepository
import com.kpi.maydanchik.data.user.UserRepository
import com.kpi.maydanchik.di.ActivityScope
import com.kpi.maydanchik.di.app.ApplicationComponent
import com.kpi.maydanchik.screens.main.MainActivity
import com.kpi.maydanchik.utils.Workers
import dagger.Component

@ActivityScope
@Component(dependencies = [ApplicationComponent::class], modules = [MainActivityModule::class])
interface MainActivityComponent {
    fun inject(activity: MainActivity)

    fun serviceProvider(): ServiceProvider

    fun workers(): Workers

    fun preferencesRepository(): PreferencesRepository

    fun userRepository(): UserRepository
}
package com.kpi.maydanchik.screens.main.di

import com.kpi.maydanchik.core.network.ServiceProvider
import com.kpi.maydanchik.data.events.ApiEventsRepository
import com.kpi.maydanchik.data.events.EventsRepository
import com.kpi.maydanchik.data.events.EventsService
import com.kpi.maydanchik.data.preferences.PreferencesRepository
import com.kpi.maydanchik.di.ActivityScope
import com.kpi.maydanchik.di.FragmentScope
import com.kpi.maydanchik.screens.main.MainPresenter
import com.kpi.maydanchik.screens.main.domain.MainInteractor
import com.kpi.maydanchik.utils.Workers
import dagger.Module
import dagger.Provides

@Module
class MainActivityModule {

    @ActivityScope
    @Provides
    fun provideMainPresenter(interactor: MainInteractor) = MainPresenter(interactor)

    @ActivityScope
    @Provides
    fun provideMainInteractor(preferencesRepository: PreferencesRepository,
                              eventsRepository: EventsRepository,
                              workers: Workers) =
            MainInteractor(preferencesRepository, eventsRepository, workers)
}
package com.kpi.maydanchik.di.app

import android.content.Context
import android.content.SharedPreferences
import com.kpi.maydanchik.MaydanchikApp
import com.kpi.maydanchik.core.network.RetrofitServiceProvider
import com.kpi.maydanchik.core.network.ServiceProvider
import com.kpi.maydanchik.core.network.TokenInterceptor
```

Текст мобільного додатку

```
import com.kpi.maydanchik.data.events.ApiEventsRepository
import com.kpi.maydanchik.data.events.EventsRepository
import com.kpi.maydanchik.data.events.EventsService
import com.kpi.maydanchik.data.preferences.PreferencesRepository
import com.kpi.maydanchik.data.preferences.PrefsRepository
import com.kpi.maydanchik.data.user.ApiUserRepository
import com.kpi.maydanchik.data.user.UserRepository
import com.kpi.maydanchik.data.user.UserService
import com.kpi.maydanchik.di.ActivityScope
import com.kpi.maydanchik.utils.Workers
import dagger.Module
import dagger.Provides
import io.reactivex.android.schedulers.AndroidSchedulers
import io.reactivex.schedulers.Schedulers
import javax.inject.Singleton

@Module
class ApplicationModule(val application: MaydanchikApp) {
  companion object {
     const val PREFERENCES = "PREFERENCES"
     const val BASE_URL = "http://46.101.218.209/"
  }

  @Provides
  @Singleton
  fun provideApplicationContext(): Context = application

  @Provides
  @Singleton
  fun provideWorkers(): Workers = Workers(Schedulers.io(), AndroidSchedulers.mainThread())

  @Provides
  @Singleton
  fun provideServiceProvider(tokenInterceptor: TokenInterceptor): ServiceProvider =
        RetrofitServiceProvider(BASE_URL, tokenInterceptor)

  @Provides
  @Singleton
  fun provideTokenInterceptor(prefRepository: PreferencesRepository): TokenInterceptor =
        TokenInterceptor(prefRepository)

  @Provides
  @Singleton
  fun providePreferencesRepository(preferences: SharedPreferences): PreferencesRepository =
        PrefsRepository(preferences)

  @Provides
  @Singleton
  fun provideSharedPreferences(): SharedPreferences =
        application.getSharedPreferences(PREFERENCES, Context.MODE_PRIVATE)

  @Provides
  @Singleton
  fun provideUserService(serviceProvider: ServiceProvider) = serviceProvider.createService(UserService::class.java)

  @Provides
  @Singleton
  fun provideUserRepository(userService: UserService): UserRepository = ApiUserRepository(userService)

  @Singleton
  @Provides
  fun provideEventRepository(service: EventsService): EventsRepository = ApiEventsRepository(service)

  @Singleton
  @Provides
  fun provideEventService(serviceProver: ServiceProvider) = serviceProver.createService(EventsService::class.java)
}
package com.kpi.maydanchik.di.app

import com.kpi.maydanchik.MaydanchikApp
import com.kpi.maydanchik.core.network.ServiceProvider
import com.kpi.maydanchik.data.events.EventsRepository
import com.kpi.maydanchik.data.preferences.PreferencesRepository
import com.kpi.maydanchik.data.user.UserRepository
import com.kpi.maydanchik.utils.Workers
```

Текст мобільного додатку

```
import dagger.Component
import javax.inject.Singleton

@Singleton
@Component(modules = [ApplicationModule::class])
interface ApplicationComponent {

  fun inject(app: MaydanchikApp)

  fun serviceProvider(): ServiceProvider

  fun preferencesRepository(): PreferencesRepository

  fun eventRepository(): EventsRepository

  fun workers(): Workers

  fun userRepository(): UserRepository
}
package com.kpi.maydanchik.core.base

import android.content.Intent
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.kpi.maydanchik.MaydanchikApp
import com.kpi.maydanchik.screens.login.LoginActivity

abstract class BaseActivity : AppCompatActivity(), BaseView {

  private var loadingDialog: LoadingDialog? = null

  protected fun getComponent() = (application as MaydanchikApp).applicationComponent

  override fun hideLoading() {
    loadingDialog?.dismiss()
  }

  override fun showError(error: Throwable) {
    Toast.makeText(this, error.message, Toast.LENGTH_LONG).show()
  }

  override fun showError(error: String?) {
    error?.let { Toast.makeText(this, error, Toast.LENGTH_LONG).show() }
  }

  override fun showLoading() {
    loadingDialog = LoadingDialog()
    loadingDialog?.show(supportFragmentManager, LoadingDialog::class.java.simpleName)
  }

  override fun showNetworkErrorDialog() {
    //todo
  }

  override fun showInitScreen() {
    val intent = Intent(this, LoginActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
    startActivity(intent)
  }

}
package com.kpi.maydanchik.core.base

import io.reactivex.disposables.CompositeDisposable

abstract class BaseInteractor {

  protected val disposables = CompositeDisposable()

  open fun unsubscribe() = disposables.clear()
}
package com.kpi.maydanchik.core.base

import android.os.Bundle
import androidx.fragment.app.Fragment
```

Текст мобільного додатку

```
abstract class BaseFragment : Fragment(), BaseView {

  val safeActivity: BaseActivity
    get() {
      return activity as BaseActivity
    }

  val safeArguments: Bundle
    get() {
      return arguments ?: throw NullPointerException("Arguments are null.")
    }

  override fun hideLoading() {
    safeActivity.hideLoading()
  }

  override fun showError(error: Throwable) {
    safeActivity.showError(error)
  }

  override fun showInitScreen() {
    safeActivity.showInitScreen()
  }package com.kpi.maydanchik.core.base

interface BaseView {

  fun hideLoading()

  fun showLoading()

  fun showError(error: Throwable)

  fun showError(error: String?)

  fun showNetworkErrorDialog()

  fun showInitScreen()
}

  override fun showLoading() {
    safeActivity.showLoading()
  }

  override fun showNetworkErrorDialog() {
    safeActivity.showNetworkErrorDialog()
  }

  override fun showError(error: String?) {
    safeActivity.showError(error)
  }
}
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 android:layout_width="match_parent"
 android:layout_height="match_parent">

 <ImageView
   android:id="@+id/backIv"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:padding="16dp"
   android:src="@drawable/ic_arrow_black"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent" />

 <TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Contacts"
   android:textColor="@android:color/black"
   android:textSize="20sp"
   app:layout_constraintBottom_toBottomOf="@id/backIv"
   app:layout_constraintLeft_toLeftOf="parent"
```

Текст мобільного додатку

```
 app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="@id/backIv" />

  <TextView
    android:id="@+id/addressTv"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:layout_marginBottom="32dp"
    android:text="22 Yangelya st, Kyiv, Ukraine"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

  <ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/map"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>



package com.kpi.maydanchik.screens.login

import android.content.Intent
import android.os.Bundle
import com.kpi.maydanchik.MaydanchikApp
import com.kpi.maydanchik.R
import com.kpi.maydanchik.core.base.BaseActivity
import com.kpi.maydanchik.screens.login.di.DaggerLoginComponent
import com.kpi.maydanchik.screens.login.di.LoginModule
import com.kpi.maydanchik.screens.main.MainActivity
import com.kpi.maydanchik.utils.setClickability
import kotlinx.android.synthetic.main.activity_login.*
import javax.inject.Inject

class LoginActivity : BaseActivity(), LoginView {

   private val component by lazy {
      DaggerLoginComponent.builder()
            .applicationComponent((application as MaydanchikApp).applicationComponent)
            .loginModule(LoginModule(this))
            .build()
   }

   private var forLogin = true

   @Inject
   lateinit var presenter: LoginPresenter

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_login)
      component.inject(this)
      initViews()
   }

   override fun onDestroy() {
      super.onDestroy()
      presenter.onDetach()
   }

   private fun initViews() {
      loginButton.setOnClickListener {
         if (forLogin)
            presenter.login(emailEt.text.toString(), passwordEt.text.toString())
         else
            presenter.register(emailEt.text.toString(),
                  usernameEt.text.toString(),
                  firstnameEt.text.toString(),
                  lastnameEt.text.toString(),
```

Текст мобільного додатку

```
            passwordEt.text.toString())
        }

        registerEt.setOnClickListener { changeScreenState() }
        loginEt.setOnClickListener { changeScreenState() }
    }

    private fun changeScreenState() {
        if (forLogin) motionLayout.transitionToEnd()
        else motionLayout.transitionToStart()

        forLogin = !forLogin
        registerEt.setClickability(forLogin)
        loginEt.setClickability(!forLogin)
        loginButton.text = if (forLogin) getString(R.string.hint_login) else getString(R.string.hint_register)
    }

    private fun setRegisterFieldsClickable() {
        firstnameEt.setClickability(!firstnameEt.isClickable)
        lastnameEt.setClickability(!lastnameEt.isClickable)
        usernameEt.setClickability(!usernameEt.isClickable)
    }

    override fun showAuthError() {
        showError("You have entered wrong email or password.")
    }

    override fun showEmailError() {
        emailEt.error = "Invalid email"
    }

    override fun showPasswordError() {
        passwordEt.error = "Invalid password"
    }

    override fun showUsernameError() {
        usernameEt.error = "Invalid username"
    }

    override fun showFirstnameError() {
        firstnameEt.error = "Invalid name"
    }

    override fun showSecondnameError() {
        lastnameEt.error = "Invalid name"
    }

    override fun startMainActivity() {
        val intent = Intent(this, MainActivity::class.java)
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
        startActivity(intent)
    }
}
package com.kpi.maydanchik.screens.login

import com.kpi.maydanchik.core.base.BasePresenter
import com.kpi.maydanchik.screens.login.domain.LoginInteractor
import retrofit2.HttpException
import java.net.HttpURLConnection

class LoginPresenter(private val interactor: LoginInteractor) : BasePresenter<LoginView>(interactor) {

    fun login(email: String, password: String) {
        mvpView?.showLoading()
        interactor.login(email, password, {
            mvpView?.hideLoading()
            mvpView?.startMainActivity()
        }, ::onLoginError)
    }

    fun register(email: String, username: String, firstname: String, lastname: String, password: String) {
        mvpView?.showLoading()
        interactor.register(email, username, firstname, lastname, password, {
            mvpView?.hideLoading()
            mvpView?.startMainActivity()
```

```
        }, ::onInvalidValueError)
    }

    private fun onLoginError(error: Throwable) {
        if (error is HttpException) {
            when (error.code()) {
                HttpURLConnection.HTTP_UNAUTHORIZED -> {
                    mvpView?.hideLoading()
                    mvpView?.showAuthError()
                }
            }
        } else onInvalidValueError(error)
    }

    private fun onInvalidValueError(error: Throwable) {
        mvpView?.hideLoading()
        when (error) {
            is LoginInteractor.InvalidPasswordException -> mvpView?.showPasswordError()
            is LoginInteractor.InvalidEmailException -> mvpView?.showEmailError()
            is LoginInteractor.InvalidFirstNameException -> mvpView?.showFirstnameError()
            is LoginInteractor.InvalidSecondNameException -> mvpView?.showSecondnameError()
            is LoginInteractor.InvalidUsernameException -> mvpView?.showUsernameError()
            else -> onError(error)
        }
    }
}
```

ДОДАТОК В

Розробка спеціалізованого Інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51143_19Б

Аркушів 9

Київ 2019

АНОТАЦІЯ

Даний додаток містить опис програми для мобільного пристрою на платформі Android. Створений мобільний додаток виконує такі задачі:

— введення даних для авторизації;

— завантаження даних з сервера;

— обробка ти вивід даних с сервера;

— відправлення даних на сервер .

Програма взаємодіє з користувачем завдяки графічною інтерфейсу користувача

При розробці мобільного додатку було використано мову Kotlin та Java у середовищі програмування Android Studio з використанням системи автоматичної збірки проекту Gradle.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис системи бронювання яка була розроблена під платформу Android. У додатку Б міститься програмний код головних модулів розроблюваної системи.

Мобільний додаток працює на операційній системі Android 5.1 Lollipop, додаткових встановлень на мобільний пристрій не потрібно.

При розробці мобільного налаштунку було використано:

- Середовище розробки Android Studio;
- Система автоматичної збірки проекту Gradle.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблені компоненти виконують завдання бронювання спортивних подій, а саме бронюють певний інтервал часу на певному спортивному майданчику.

Розроблений додаток можна інтегрувати та використовувати у реальному житті з реальними спортивними майданчиками з е-сервісами.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для реалізації мобільного додатку знадобилось використання багатьох бібліотек та фреймворків. В проект було впроваджено архітектурний шаблон MVP.

В проекті було використано фреймворк Dagger 2 для впровадження залежностей та впровадження Inversion of Control

Для роботи з інтернетом використано бібліотеку Retrofit.

Для асинхронності було використано бібліотеку RxJava.

При запуску систем користувач повинен авторизуватися, щоб користуватися основним функціоналом. Після того як користувач авторизується, йому буде надана можливість бронювання місця у події та можливість створення власної події.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та досягнення високої ефективності роботи створеного додатка було використано середовище розробки Android Studio, набір інструментів розробника Android SDK та мову Kotlin.

Розроблений додаток працює в операційній системі Android 5.1 та новіше. Не потребує встановлення додаткових програм чи інструментів.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблена система потребує інсталяції на мобільному пристрої під керівництвом операційної системи Android. Необхідно запустити файл maydanchik.apk на мобільному пристрої, після чого операційна система встановить програму та запустить її.

Після запуску користувач отримає доступ до всього функціоналу додатка.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розроблених додатків є інформація яка зчитується з EditText віджетів.

Вхідні дані можуть мати вигляд цілого числа або строки.

Вихідними даними є дані з сервера, вони можуть приходити у вигляді JSON-об’єктів, у вигляді цілочисельних значень та строк.